c.2

LBL-10409
UC-32

NRCC NATIONAL RESOURCE FOR COMPUTATION IN CHEMISTRY

# ATTACHED SCIENTIFIC PROCESSORS FOR CHEMICAL COMPUTATIONS: A REPORT TO THE CHEMISTRY COMMUNITY

Neil S. Ostlund

January 1980

**LAWRENCE BERKELEY LABORATORY
UNIVERSITY OF CALIFORNIA**

Prepared for the U.S. Department of Energy under Contract W-7405-ENG-48 and for the National Science Foundation under Interagency Agreement CHE-7721305

National Resource for Computation in Chemistry

# Attached Scientific Processors for Chemical Computations:

A Report
to the
Chemistry Community

Neil S. Ostlund
Department of Chemistry
University of Arkansas
Fayetteville, AR 72701

INDEX

## 1.0 Purpose and Scope of this Report

The demands of chemists for computational resources is well known and has been amply documented[1]. The best and most cost-effective means of providing these resources is still open to discussion, however. This report, sponsored by the National Resource for Computation in Chemistry (NRCC), surveys the field of attached scientific processors ("array processors") and attempts to indicate their present and possible future use in computational chemistry. Array processors, for example, the AP-120B produced by Floating Point Systems, Inc., have the possibility of providing very cost-effective computation. Definitive information concerning array processors has not, however, been generally available to computational chemists, nor has there been a general appreciation within the community of the good and bad characteristics of this mode of computation. This report attempts to provide information which will assist chemists who might be considering the use of an array processor for their computations. It describes the general ideas and concepts involved in using array processors, the commercial products that are available, and the experiences reported by those currently using them. In surveying the field of array processors, the author makes certain recommendations regarding their use in computational chemistry and recommends ways in which NRCC might play a role in advancing this use. The opinions expressed, however, are solely those of the author and in no way reflect NRCC policy.

In conjunction with this study a small 1-1/2 day meeting, "Array Processors for Chemical Computations," was held at the NRCC on July 20-21, 1979. In addition to myself, the NRCC staff, and a small number of interested individuals in the local area, six people with expertise in the area were invited to describe their experience with array processors and to relay to those in attendence their impression of the current and future status of array processors for chemical computations. This report abstracts many of the points brought forward in that meeting. The author acknowledges, if not explicitly, the assistance of those in attendance and many others who have contributed information about array processors. In the final analysis, however, the opinions expressed are those of the author and apologies are made to any group or individual misrepresented.

Section 2 gives some background on the advancing state of hardware technology and parallelism in architectures. Section 3 describes some of the general concepts relevant to understanding how most array processors function. Section 4 explicitly describes a few of the available commercial products - those most relevant to computational chemistry. Section 5 describes the experience of scientists using an array processor for applications directly related to those of computational chemistry. Section 6 includes a general discussion on the applicability of array processors to chemical problems. Section 7 reviews the available literature on array processors. Section 8 introduces possible roles the NRCC might play in advancing the state of chemical computation via array processing. Finally, Section 9 presents the conclusions and recommendations of this report.

## 2.0 Advancing Hardware Technology

Computers are built from registers, adders, multiplexers, decoders, etc., which in turn are built from logic gates (NOT, AND, OR, etc.), which are in turn built from transistors on the surface of a silicon chip[2]. The level of integration, i.e., the complexity of circuitry placed on a single chip, has been increasing dramatically and will continue to do so. Developments in the semiconductor industry will have a profound influence on every aspect of science and technology. In view of the reliance of computational chemistry on computers, it is important that computational chemists be aware of these advances in order to take best advantage of current and future developments in microelectronics and associated computer hardware.

Integrated circuits can be characterized by their density (number of gates per unit area) and by their logic family (the type of transistor used and the method of interconnecting transistors). The three most significant technologies (logic families), in order of increasing density but decreasing speed, are ECL (emitter-coupled logic), TTL (transistor-transistor logic), and N-MOS ( N-channel metal oxide semiconductor). Each of the three technologies is characterized by a switching speed[3]: the delay in the output of a single logic gate subsequent to a change in its input. A rough estimate of the switching speeds of ECL, TTL and N-MOS is 1, 5 and 50 nanoseconds, respectively. Gates connected in a serial fashion, of course, accumulate this delay. Most current computers use TTL logic, very high speed circuits use ECL, and microprocessor chips use N-MOS technology.

The densities of integrated circuits vary from a few tens of transistors per chip (SSI, small-scale integration) to tens of thousands of transistors per chip (VLSI, very-large-scale integration). The level of integration available varies in the order N-MOS $>$ TTL $>$ ECL. While the densities of each of the three technologies is increasing, the most dramatic increases have been with N-MOS and it is now possible, using this technology, to place complete central processing units (CPU's) on a single chip. An announced product, the 68000 microprocessor from Motorola[4], will have of the order of 90,000 transistors on a single chip. Current N-MOS memory chips have 64K (K=1024) bits of memory. The number of transistors in these chips is perhaps only an order of magnitude less than that in many large mainframe CPU's. In the next decade these densities can be expected to increase by perhaps two orders of magnitude[5]. Thus, in spite of expected advances in chemical theory (and experiment) and advances in computational methods, algorithms, and efficient application programs, it is possible that the most significant development determining the course of computational chemistry in the next 10 years will be the continuing revolution in microelectronics.

Independent of the level of integration, the production cost of a single chip is, at most, a few dollars. If the large development cost can be paid off by a mass market, relatively sophisticated computing power will be available to scientists at a cost compatible with the budget of an individual researcher. It seems clear that a desktop computer with the processing power of, for example, an IBM 370/145 will be available for $10,000 in the not too distant future. As hardware costs continue to drop, the personal computer will proliferate.

While densities will continue to increase, the speed of logic circuits can not be expected to improve in a similarly dramatic way. Some improvement can be expected in N-MOS and there already exist memory chips with access times under 50 nanoseconds. The Josephson junction[6] offers some hope of obtaining switching speeds in the picosecond range but these require liquid Helium temperatures and computers formed from them are a considerable distance down the road. In searching for larger and larger processing power, effort appears to be best placed in greater and greater degrees of parallelism. The usual serial computer will become cheaper and cheaper but it cannot be expected to become significantly faster. While a not insignificant amount of computing power will be widely distributed to individual chemists in the near future, a search should, and surely will, continue for cost-effective ways of solving larger and larger computational problems. The computations that chemists would like to perform are essentially infinite. Quantum chemists, as an example, will always seek more and more accurate calculations on larger and larger systems. While proliferation of today's serial computer will result in solutions to most of the problems now being tackled, only new parallel architectures will enable chemists to expand their horizon and seek solutions to problems previously considered intractable.

The future of high performance computers is in parallel computation. This is not necessarily a recent concept; the simultaneous processing of all bits in a computer word is one form of parallel processing, as is the simultaneous execution of an instruction and the fetch from memory of a subsequent one (IBM 7094). Current machines like the CDC 7600 and Cray-1 require a significant degree of parallelism in their architecture in order to achieve their high performance. With dropping hardware costs, the options available to designers seeking high performance is very large. Perhaps in the distant future, one parallel architecture will dominate; in the meantime, there is likely to be large differences in "supercomputers", as one moves away from the simple Von Neumann serial machine. Present architectures largely hide any degree of parallelism from the user. Unlike these past improvements in computer hardware, which conferred increased speed without requiring major changes in programming practice, it is not clear that the full benefits of parallel processing can be realized without drastic changes in operating systems, programming languages, algorithms, and the very way we approach a computation[7].

In spite of the multiplicity of parallel architectures that can and have been envisioned, it is probably still useful to classify[8] parallel machines according to parallelism in the instruction stream or data stream. The usual serial computer can be thought of as a control unit sequencing a processing unit (arthmetic and logic unit, ALU). The control unit interprets a stream of instructions and directs the ALU to process a stream of data. The combination of a control unit and processing unit is the CPU. Machines of this simple type may be referred to as single instruction stream, single data stream (SISD) computers. In this context, the most general architecture is that of the multiple instruction stream, multiple data stream (MIMD) computer. With this architecture one has, in essence, individual computers, or at least individual CPU's, which communicate with each other and (hopefully) cooperate in the solution of a single task. If the computers are loosely coupled and communicate by a relatively low speed serial line, the architecture is termed a network. The effective use of a network for the solution of a single task requires a task which can be broken into sub-tasks (processes) which are nearly disjoint and communicate very infrequently with each other. To date, networks have not been used to solve large computational problems, but normally are used to transfer data, messages, mail, etc.

If the processors of a MIMD computer are tightly coupled and communicate via a high speed common memory, the architecture is referred to as a multiprocessor. The use of auxiliary I/O processors is one example of a multiprocessor. C.mmp[9] multiprocessor in the computer science department of Carnegie-Mellon University has 16 memory units connected to 16 PDP-11's by a crosspoint switching device, such that any processor can access any memory unit. Communication of data, messages, etc., between processors occurs via their common memory. Although dual processors and multiprocessors with a small ($\sim$ 3-4) degree of parallelism (apart from I/O processors) are available commercially, no multiprocessors with a large number of processors, suitable for large-scale scientific computation, have yet appeared. The driving force behind multiprocessor architectures is the potential cost-effectiveness of connecting hundreds or even thousands of inexpensive microprocessors. This is an active area of research, as exemplified by Cm* (50 LSI-11 microcomputers in parallel, again, in the computer science department of Carnegie-Mellon University[9]), but commercial MIMD architectures, applicable to large scientific calculations, are still a way off. The prospect of using multiprocessors for the computations of chemistry has recently been discussed by the author[9]. If, as many computer scientists foresee, this is the ultimate parallel architecture, huge reinvestments in algorithms and software will be required. Serial

FORTRAN and present computational methods are simply not adequate for efficient multiprocessing.

At a lower level of parallelism are single instruction stream, multiple data stream (SIMD) computers. The canonical example is ILLIAC IV[10]. This machine has a single control unit which broadcasts decoded instructions to 64 processing units (ALU's). Each processing unit has a small amount of local memory which holds data and each processor executes exactly the same instruction, although on different data. By a mask, it is possible, however, for any processor to idle rather than actually execute an instruction. As opposed to the "array processors" which are the subject of this report, ILLIAC IV is a true array processor and its 64 processing units are arranged as an 8x8 array. Its architecture is best suited to calculations which manipulate arrays of data. For code such as

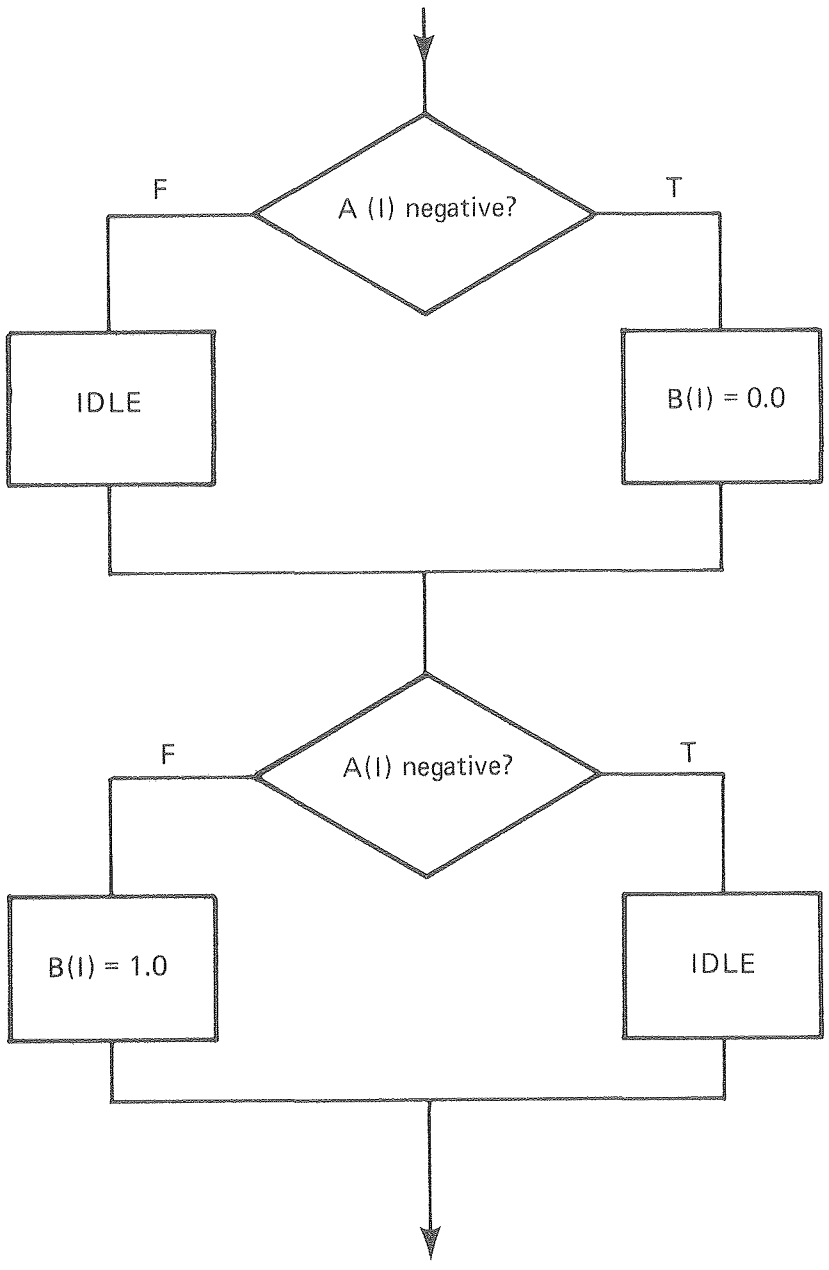
```
          DO  10  I= 1,64
     10   C(I)=  A(I) + B(I)
```

each of the processing units can operate simultaneously on its own component of the data, since the same instruction can be executed by each. If, however, the code includes branching, such as the following two-way branch,

```
          DO  20  I= 1,64
          IF  (A(I).LT.0) GO TO 10
          B(I) = 1.0
          GO TO 20
     10   B(I) = 0.0
     20   CONTINUE
```

then, since each processing unit must either be idle or execute the common instruction, the parallelism is reduced to 32 as illustrated by Figure 1. With inherently serial code or multi-way branching, the speed of ILLIAC IV can be slowed down to that of a single processing unit. A SIMD architecture thus achieves high throughput only for well-structured problems which manipulate arrays of data, with very few control statements (GO TO, computed GO TO, and IF statements). The multiplication or diagonalization of large arrays are examples of problems which a SIMD machine could be expected to execute efficiently.

The remaining machines which we wish to discuss are not arrays of processors like ILLIAC IV but, perhaps, processors of arrays. That is, like ILLIAC IV, they perform best for problems involving the manipulation of well-structured arrays of data. Current examples are the Cray-1[11] and the attached scientific processors ("array processors") which are the subject of this report. Although no complete generalization is possible, they do not normally perform to their limit for other than long vector operations. In addition to its scalar instructions, the Cray-1 has vector instructions such that VMULT, for example, can initiate the multiplication $a_i * b_i$



XBL 801 - 20

Figure 1. Two-way branching with ILLIAC IV. A common instruction is broadcast to each of 64 processing units and each processing unit must either execute the instruction or rema idle. Processing units which have A(I) negative for their value of I (in local memory) must idle for the bottom assignment instruction, while those which have A(J) positive must idle for the top assignment instruction. All processing units must execute both IF statements.

of up to 64 pairs of elements (i=1,2,...64) from the vectors $\underset{\sim}{a}$ and $\underset{\sim}{b}$. The Cray-1 is a vector machine. The array processors of this report attach to a host minicomputer or mainframe. They normally are used such that, on instructions from the host, a subroutine inside the array processor executes a vector or matrix operation on data transferred from the host to the array processor. The reason why the Cray-1 and array processors execute most efficiently with vector operations is usually associated with pipelines for floating point arithmetic. Pipelines will be described in the next section. The basic idea is that, like an assembly line, discrete operations are performed, at each segment of the pipeline, on data moving through the pipeline. A new result (a floating point multiply, for example) can emerge from the pipeline in the execution time of a segment, but only if the pipeline is kept full. Performing vector operations on long vectors allows a pipeline to remain full.

The Cray-1 and other array processors are very different from ILLIAC IV. Nevertheless, since, at least on the programming level, they execute a single instruction which manipulates vectors (multiple data), they are sometimes classified with true SIMD machines. At a certain programming level they have only a single serial instruction stream, quite unlike multiprocessors. It is thus possible to generate FORTRAN compilers for them, and they perhaps provide the least perturbation to serial computation.

## 3.0 Array Processor Fundamentals

Although we believe it is sometimes useful to attempt to classify computer architectures, as we have done in the last section, every architecture has its own unique characteristics. It is thus not possible to give a uniform and simple definition of what have come to be known as array processors, and later it will be necessary to give reasonably detailed descriptions of individual products. A number of products, however, have certain characteristics in common and in this section we give a pedagogical description of some of the major architectural features.

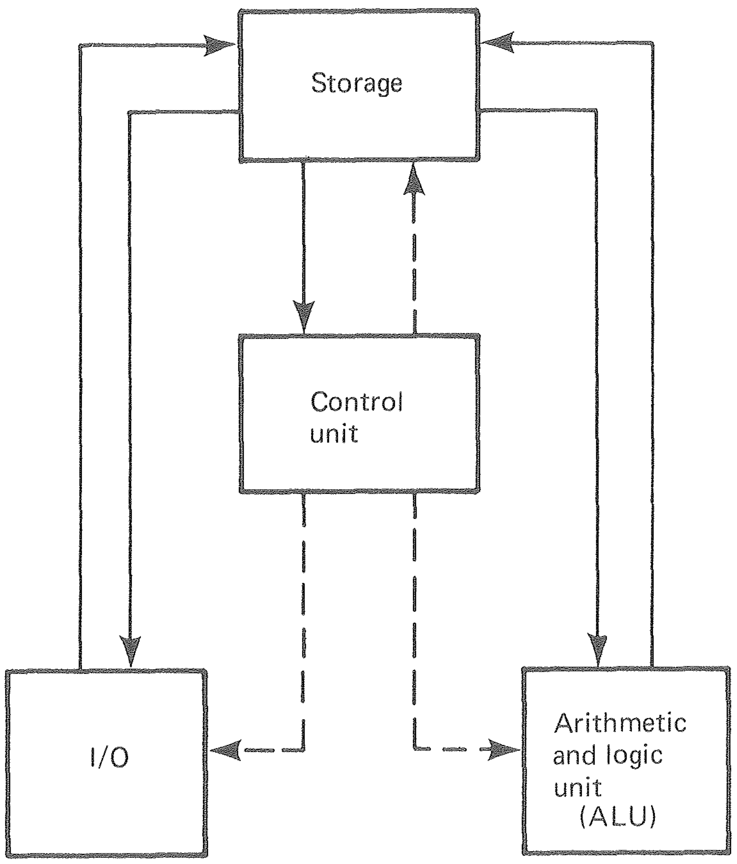
The origin of array processors is in signal processing, such as that involved in radar detection of aircraft, speech and image processing, and geological exploration. In these applications, one generally requires a dedicated computer capable of processing large amounts of data obtained from analog to digital conversion of "signals". The most common operation required is the fast Fourier transform (FFT). These signal processing applications require extensive arithmetical operations above and beyond the capabilities of the usual dedicated minicomputer. These applications do not require large precision, and in some cases not even floating point arithmetic, but they do require very fast multiplication and addition. In response to this need, there arose relatively inexpensive signal processing "boxes" which could be added on to a minicomputer, communicating with their host as a peripheral device.

The host provided the general capability of any minicomputer, but the computationally intensive operations could be performed in the attached signal processor, by passing data to it and subsequently collecting results from it.

Almost as if by accident, at least one of these add-on boxes, that produced by Floating Point Systems, Inc., has proven to be a powerful scientific "number cruncher". Thus arose the attached scientific processor with the unfortunate name "array processor". The general characteristics of an array processor are that it is capable of high performance arithmetic operations, attaches to an existing computer, and is most effective when performing operations on long arrays of data passed to it by its host. In addition, it is assumed to be relatively inexpensive. As with every rule there are exceptions. The host can be a minicomputer (usually) or a mainframe computer (sometimes). The 8 million dollar Cray-1 could be called an attached scientific processor since it also requires a host, commonly a CDC 7600. In at least one case, the Eclipse S/130 AP, the array processor is integrated with its host and not attached at all. The integral array processor would appear to be a likely prospect for the future. Unfortunately, if they include floating point operations, most current array processors use only a 32-bit floating point word. These 6 decimal digits are not sufficient for many of the calculations of computational chemistry. The major considerations in choosing an array processor are the precision of its floating point operations, its inherent speed for the type of operations that will be performed, the size of program and data memory that are available, the speed at which data can be transferred to and from the host (including all overhead), the ease of programming and the software (operating systems, compilers, assemblers, math library, etc.) available, the availablity of an interface to particular hosts, the availability and support of peripherals, for example, disks, that might be attached directly to the array processor, and of course, the cost.

## 3.1 Microcoding

A number of array processors can, or even must, be microcoded by the user. Since this concept is not generally familiar to chemists, a brief discussion of the concepts involved is given here. A quite readable introduction to microprogramming is available in references 12 and 13. The usual serial machine is shown in Figure 2. Storage in this case includes main memory as well as internal registers. The solid lines are data buses (data paths) for the movement of data. The dotted lines are control buses (control lines) for controlling read and write of storage, the operation of the arithmetic and logic unit, etc. A bus is simply one or more wires used to pass electrical information (voltages). The ALU performs addition, subtraction, logical operations, shifting, etc. In a hard-wired control unit, such as that of the CDC 6600, a machine



XBL 801 - 23

Figure 2. Functional organization of a digital computer. The control lines (dotted lines) from the control unit determine the operation of the other three units. Data passes along the data paths (solid lines).

instruction is read from the main memory into the instruction register of the control unit where the opcode (ADD, LOAD, etc.) is decoded, causing certain control lines to become active, such that the instruction is executed. Such hard-wired control units, while fast, are rather complex.

The IBM 370 and most new machines do not have hard-wired control units but instead have microprogrammed control units, although the user is not normally aware of the difference. In hard-wired control units the machine instructions (load register from memory, add memory location to register, etc.) are interpreted by the hardware. In a microprogrammed computer, on the other hand, these machine instructions are interpreted by a microprogram of microinstructions. The microinstructions are in turn interpreted by the hardware. Since microinstructions have a closer relation to the ultimate hardware of gates, buses, etc. than machine instructions, microcode is sometimes termed firmware, particularily, if it can not be easily changed. Thus a machine instruction (software) is interpreted by microcode (firmware) which in turn is interpreted by hardware.

Machine instructions can be called macroinstructions. Just as macroprogramming normally uses an assembler to translate symbolic code into binary machine instructions, a microassembler can be used to translate symbolic microinstructions into binary microcode. Microprogramming requires a relatively detailed knowledge of a computer's hardware structure and is more tedious than macroprogramming, just as normal assembly language programming tends to be tedious relative to programming in a high level language like FORTRAN. The microcode which interprets machine instructions is stored in a fast memory called the control store, which is normally a part of the control unit rather than main memory. In most machines the control store is ROM (read only memory) which is one-time programmed by the manufacturer to interpret his particular machine's instructions. The user of such a machine is generally unaware of the underlying microcode since he cannot change it nor have access to it. Other machines, however, have part or all of their microcode in RAM (random access or read/write memory) That is, these machines have writable control store. These machines thus allow the user to implement his own machine instructions. There appears to be a trend among manufacturers to allow a certain amount of user control store for implementing machine instructions beyond the standard set.

Figure 3 shows the general organization of a microprogrammable machine. It includes main memory, 16 general purpose registers (R0,...R15), an ALU and a shifter, as well as a status register flagging the result (zero or negative, for example) of arithmetic and logical operations. The lower left hand corner constitutes the control unit.

Accesses to main memory occur by moving an address into the memory address register (MAR), loading the memory data register MDR (for a write), executing the READ or WRITE command, and then loading

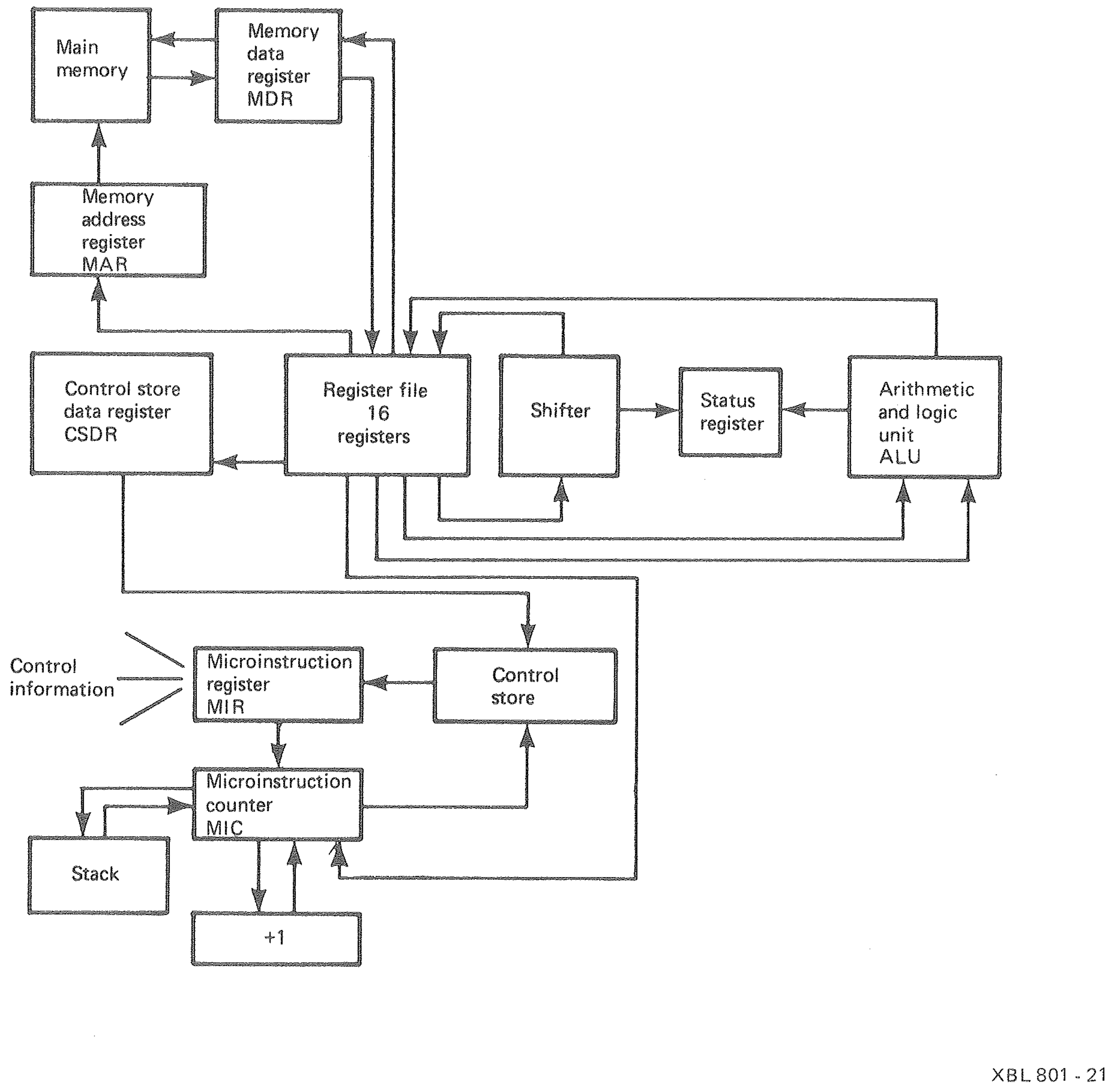

Main memory
Memory data register MDR
Memory address register MAR
Control store data register CSDR
Register file 16 registers
Shifter
Status register
Arithmetic and logic unit ALU
Control information
Microinstruction register MIR
Control store
Microinstruction counter MIC
Stack
+1
XBL 801 - 21

Figure 3. Organization of a simple microprogrammable machine

one of the general purpose registers from the MDR (for a read). Assuming the appropriate address is in R0, the three microinstructions required to read a memory location into a register R1 are:

```
MAR ← R0
READ
R1 ← MDR
```

In addition to the READ and WRITE microinstructions and the microinstructions which transfer the contents of one register to another, there will be microinstructions which perform operations on contents of the 16 general purpose registers (GPR's). For a simple micromachine these could include

```
GPR ← shifted GPR
GPR ← unary operation GPR
GPR ← GPR binary operation GPR
```

For example,

```
R3 ← R1 + R2
```

There will also be microinstructions which load a GPR with a literal value (e.g. 3).

The control unit includes the control store holding the microinstructions, the microinstruction counter (MIC) which holds the address in control store of the current microinstruction, and the microinstruction register (MIR) which holds the current microinstruction as it is decoded by the hardware. The MIC is analogous to the usual program counter (PC) of a macromachine. For normal sequential execution it needs to be incremented by 1 for each microinstruction executed. For jumps or branches it may be loaded from a field of the current instruction (MIR) or from a GPR. In addition, there may be a stack for pushing and popping the MIC upon calling and returning from a microsubroutine. The control store data register (CSDR) is used for loading microcode. In addition to the microinstructions already discussed, there must be instructions which manipulate the microinstruction counter, including conditional (depending on the bits in the status register) and unconditional branch instructions.

The fetch and execution of a machine instruction normally occurs by loading the macroprogram counter (PC), which is usually kept in one of the GPR's, into the MAR, executing the READ instructions and then loading the macroinstruction from the MDR into one of the GPR's. The opcode of the macroinstruction can be extracted by shifting and/or masking. The opcode, or some transformed version of it, then provides an address into control store for the microroutine which actually executes the macroinstruction. The operands or their addresses will normally have been previously extracted from the macroinstruction and placed in GPR's. This constitutes a very limited description of micromachines but, hopefully, it illustrates the general concept. Like macroprogramming, each microoperation could be executed sequentially. This is termed vertical microprogramming. As Figure 3 illustrates, however, there are usually a number of independent buses in a micromachine and it is possible to have different microoperations occuring in parallel. If different registers in the general purpose register file can be accessed simultaneously, as is a common situation, then the following microoperations, as an example, might all occur in parallel

$$R0 \leftarrow MDR; \quad R2 \leftarrow R1 + R2; \quad R3 \leftarrow R3 \uparrow 2; \quad CSAR \leftarrow R4$$

where $R3\uparrow 2$ indicates R3 shifted to the left by 2. Thus, one could be completing a memory read, adding two registers, shifting a register, and executing a jump in a single instruction. This is called horizontal microprogramming. The individual microoperations ($R0 \leftarrow MDR$, etc.) would occupy fields of a single "wide" microinstruction. Thus, by having an architecture with multiple data paths and independent functional units, such as the shifter and ALU of our example, it is possible to incorporate a reasonable degree of parallelism (and hence speed) at the micromachine level. Parallelism such as this is used in array processors, for example the one produced by Floating Point Systems, Inc. It might be pointed out that an alternative definition of vertical and horizontal that is in common use describes horizontal microinstructions as those which need not be decoded since individual bits determine specific microoperations.

## 3.2 Floating Point Hardware

By an appropriate combination of logic gates, it is reasonably easy to produce a circuit which is capable of adding or subtracting binary integers. Thus, most ALU's that are available as single chips or that are used in many micro- and mini-computers, are capable only of adding or subtracting integers. In some machines, integer multiplication must be implemented by software. Multiplication occurs by repeated addition and shifting much as one would multiply on paper, except in binary format. This is obviously slow, perhaps milliseconds for a floating point multiplication. Other machines implement these operations in microcode. If N-MOS is used this is still very slow. A new single chip floating point unit from Advanced Micro Devices, the AMD 9512, requires 99 microseconds for a 32-bit floating point multiplication and 874 microseconds for a 64-bit floating point multiplication. If a microprogrammed, multiple chip TTL processor is used for floating point operations, these times are much better but still slow for high performance scientific computation. To achieve high performance in floating point operations, it is necessary to have special purpose rather complex hardware floating point units built from SSI and MSI (medium scale integration) TTL, or possibly ECL, chips. Array processors provide these high performance floating point units, not generally available in minicomputers. Most minicomputer vendors offer a floating point accelerator, floating point unit, etc. as an option but these do not yet match in performance those available in array processors.

## 3.3 Pipelining

Floating point operations, even when accomplished in fast hardware, are time consuming. To speed them up further, a number of array processors and most other high performance machines, such as the Texas Instrument Advanced Scientific Computer (ASC) and the Cray-1, pipeline these operations. To illustrate this concept we will use the operation of floating point addition. A floating point number includes a mantissa (fraction) and an exponent. Although some computers use base 16, most use base 2 and we will use as an example of a floating point word, one bit for the sign of the number (0 for positive, 1 for negative), 3 bits for the exponent or power of 2, and 4 bits for the binary fraction. Thus, the word 00101011 represents $+2^{010} \times 0.1011$. This, equivalently, is $2^2 \times (2^{-1} + 2^{-3} + 2^{-4}) = 4 \times (1/2 + 1/8 + 1/16) = 2.750$ decimal. Suppose now that we wish to add this number to 0.875 decimal = 00001110 = $+2^{000} \times 0.1110$. At least four distinct steps, which must be executed sequentially, are required for this addition:

1. The smaller of the two exponents must be subtracted from the larger exponent:
   $010 - 000 = 010 = 2_{10}$
2. The mantissa of the smaller number must be shifted to the right by the number of places given by the previous subtraction:
   $0.11100000 \longrightarrow 0.00111000$
3. The new mantissa and the mantissa of the larger number must be added:

$$\begin{array}{r} 0.10110000 \\ \underline{0.00111000} \\ \hline 0.11101000 \end{array}$$

4. The final mantissa must be normalized and then rounded (or truncated) to 4 bits. The final exponent is that of the larger original number:

$$(2^{010} \times 0.1011) + (2^{000} \times 0.1110) \approx 2^{010} \times 0.1111$$

$$2.750 \quad + \quad 0.875 \quad \approx \quad 3.75$$

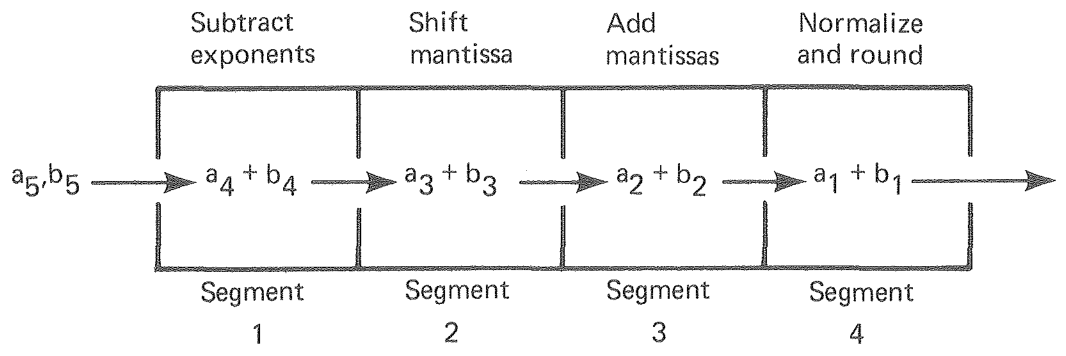
The finite precision (4 bits) gives an answer 3.75, which is different from the exact result of 3.625. The original numbers were normalized, i.e., the exponents were chosen so that the leading digit in the mantissa was a 1. In general, step 4 of the above should also include any required normalization of the result prior to rounding (or truncating), although in this example the result is already normalized.

As can be seen from the example, these steps need to be performed sequentially and the total time taken for the floating point addition will be (assuming each of the steps takes approximately the same time) four times that for an individual step. This may be too long for some requirements. Assuming that there are many numbers that we want to add, one way to speed up the addition is to form a segmented pipeline. In our case, the pipeline will have 4 segments, corresponding to each of the four operations described above, as shown in Figure 4. If we want to add the numbers $a_i + b_i$, $i = 1, 2, \ldots$, we first let $a_1$ and $b_1$ enter the pipeline at segment 1. Most computer's operations are clocked by a pulse train of a certain frequency. We can assume that the operations of each segment require one clock cycle. At the end of the first clock cycle, the exponents of $a_1$ and $b_1$ will have been subtracted and the appropriate operands can move to the second segment of the pipeline. At the same time, $a_2$ and $b_2$ can enter the pipeline at segment 1. Provided we have a continuous stream of operands $a_i$ and $b_i$ moving through the pipeline, a sum will emerge from the end of the pipeline every clock cycle. If on the other hand, only two numbers need to be added, 4 clock cycles will be required for the operands to completely move through the pipeline. Thus, a pipeline becomes effective only if a continuous stream of operands is available. This can be accomplished by vector operations. If two long vectors are to be added, the overhead of filling and draining the pipe is negligible and our pipeline for floating point addition would give a sum every clock cycle, or in 1/4 the time required for an isolated sum.

Machines with long pipelines will only perform close to their limit provided the problem can be structured in terms of vector operations. An array processor which operates on large arrays of data or performs operations on large matrices can make use the pipeline concept very effectively.

## 3.4 Synchronous and Asynchronous Devices

A single processor or a computer with a single control unit is normally a synchronous device. Events such as loading a register, activating an ALU, etc. are clocked by a pulse train and occur at predictable times. External interrupts, of course, occur asynchronously. Most SIMD machines, having a single control unit, are synchronous. ILLIAC IV, for example, operates in a lock-step



XBL 801 - 22

Figure 4. A pipeline for floating point addition. The result $c_1=a_1+b_1$ emerges from the last segment of the pipeline at the same time the operands $a_5$ and $b_5$ enter the first segment. Each segment of the pipeline performs one step in the addition of two operands.

fashion such that every one of the 64 processing elements is clocked by a global clock. Multiprocessors, on the other hand, are normally asynchronous in that each processor has its own clock and events in one processor are not in step with those in another processor. For an asynchronous multiprocessor, a totally asynchronous algorithm is preferred, if one can be found, since synchronization is costly. Synchronizing events in an asynchronous device requires flags or signals to be communicated from one processor to another and perhaps back again (hand shaking), in order to indicate when a calculation is completed, data is ready, data is required, etc. Some synchronization is, of course, inevitable but it should be minimized, since it requires a processor to remain idle. If an asynchronous algorithm is not used, having to synchronize events in an asynchronous multiprocessor, particularily if it has to be done often, is a disadvantage. On the other hand, an asynchronous multiprocessor is more flexible and not as restricted to vector operations.

Some array processors have only a single control unit and every operation is clocked in a lock-step fashion. The unit from Floating Point Systems, Inc. is one example. Alternatively, some array processors, such as those produced by CSP, Inc., are effectively multiprocessors, with more than one control unit. The unit produced by Datawest Corporation has four control units but, unlike the norm, these operate in a synchronous fashion. The four control units include only one program counter, and only one control unit, the master, is allowed to execute jump instructions. The individual units can execute different instructions, but all four must execute exactly the same number of instructions (including possible do-nothing instructions). The array processor from CSP, Inc. is more like the classical multiprocessor, in that it is an asynchronous device and individual processors must set hardware flags to synchronize global events.

## 3.5 Transfer of Data To and From a Host

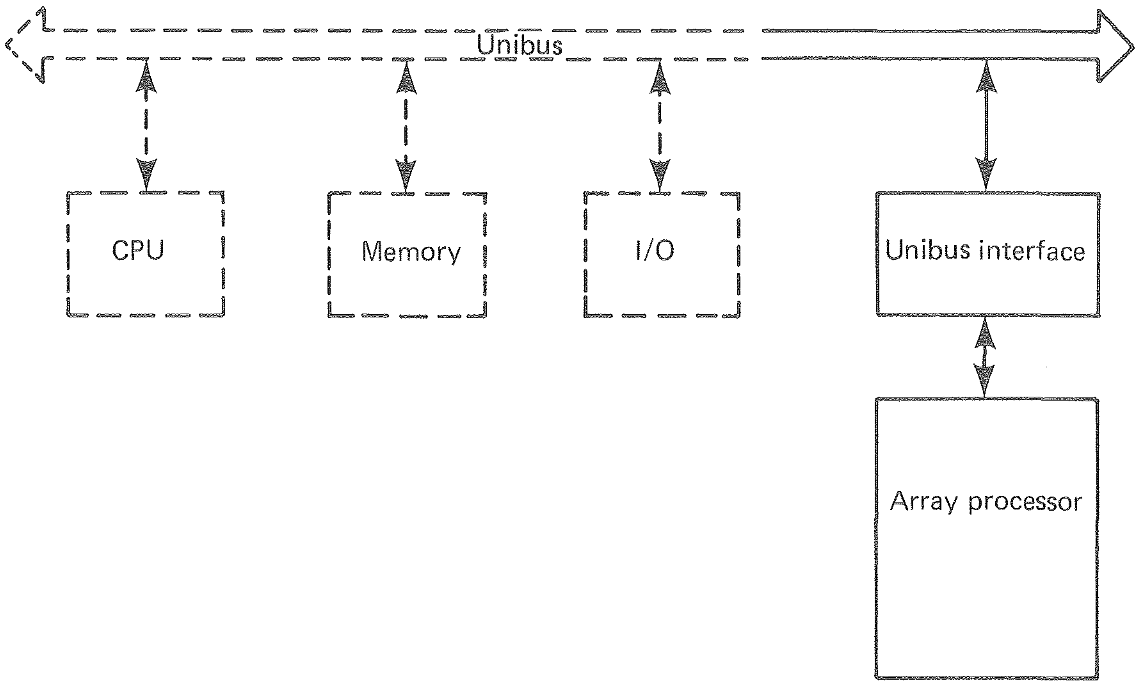
An array processor generally attaches to a host through some form of interface. A critical question in the applicability of an array processor to various computational problems is the speed at which data can be transferred between the host and the AP. Some calculations may involve very little transfer of data and have a small code which will fit entirely into the array processor. For this case, one can expect an array processor to perform well. The Monte Carlo calculations of statistical mechanics, for example, would appear to satisfy these contraints. On the other hand, if a problem requires a great deal of data, i.e., if the number of operations performed in the array processor, per fetch of a data element from the host, is small, then the speed of the interface may be extremely important. In the worst case, the array processor would effectively remain idle, waiting constantly for data to be transferred.

Two types of data transfers between a host and a peripheral device are common. Consider, as an example, the transfer of data from the host memory to an interface. The first type of transfer is programmed I/O. Here, the host would explicitly execute instructions to effect the transfer. It might, for example, read a word from memory into a register and then execute an I/O instruction to transfer the contents of the register to the interface. Alternatively, with some hosts a single instruction might effect the transfer directly. If the host was a PDP-11, these transfers would take place via the Unibus shown in Figure 5. Programmed I/O for block data transfers requires the continuous fetching and execution of instructions by the host and is slow compared to the second alternative, which is direct memory access (DMA). With DMA, the interface is "intelligent." It suspends the host CPU, captures the bus (Unibus, in this case), and directly transfers blocks of data from the host memory to the interface. In the same way that transfers from the host to the interface can involve either programmed I/O or DMA, transfers from the interface to the array processor may also involve either programmed I/O or DMA. Not all array processors have this complete flexibility.

The rate of data transfer depends on whether it is programmed or DMA transfer, and the inherent speed of the bus. In addition, if transfers are initiated by FORTRAN calls in the host, as is usual, there can be a large overhead in initiating the transfer--in both the manufacturer's supplied routines which call the host operating system and in the host operating system itself. These overheads can be many milliseconds and are deadly if many small data transfers must be made. Given a particular application, it is important to be sure that data transfers can take place at a rate which will keep the array processor busy.

## 4.0 Current Commercial Products

Since there are very few unifying architectural features among array processors it is necessary to describe individual products and their characteristics. There is a considerable number of products on the market but most of these are signal processors, have only integer arithmetic, have limited precision, or are otherwise not significant for high performance scientific calculation. Mention will only be made of those products which it is thought might be of greatest interest to computational chemists. The distinction is based mainly on the precision of floating point computations. No attempt is made to review array processors with a floating point word equal to or smaller than 32 bits.



XBL 801 - 24

Figure 5. Array Processor Attached to a Unibus

## 4.1 Floating Point Systems, Inc., Beaverton, OR 97223

The market for scientific number crunching with array processors is dominated by Floating Point Systems. Their AP-120B, which attaches to minicomputers, and their AP-190L, which attaches to mainframes like the IBM 370, are identical except for the host interface. A new product, the FPS-100 is essentially identical to the other two except that it uses cheaper, slower, and less power consuming LS-TTL (low-power Schottky) chips rather than S-TTL (high-speed Schottky) chips. It has a cycle time of 250 nanoseconds rather than the 167 nanoseconds of the AP-120B and AP-190L. It is an OEM (original equipment manufacturers) product, however, and is only sold in quantities of 20 or more.

A wide variety of interfaces are available for the Floating Point Systems' (FPS) product, with the Unibus interface being the most common. The Unibus has an absolute maximum transfer rate of 1.5 Mbytes/sec. Presently, to interface to a VAX-11/780, the AP-120B must attach to the Unibus, although it probably will eventually interface directly to the much faster synchronous backplane interconnect (SBI). Digital Equipment Corporation (DEC) is developing an interface for the SBI which would facilitate attaching an AP-120B to the SBI. It is unlikely that this faster interface will be available prior to 1980. FPS has indicated an intent to develop an interface for the LSI-11 bus. This is of considerable interest since it would allow a relatively inexpensive microcomputer, such as the PDP-11/23, to act as a host. FPS has sold a number of FPS-100's to First Data Corporation of Westmont, Illinois who have indicated that they will package a PDP-11/23 and an FPS-100. This may be an economical alternative for users not having or requiring a more sophisticated host.

Some of the interfaces that exist are to the IBM 370, DEC 10, UNIVAC, PDP-11, PRIME, INTERDATA, HEWLETT-PACKARD, HARRIS, and ECLIPSE. It is advisable to consult FPS about interfaces for particular hosts. It would appear that they are sometimes willing to develop new interfaces when there is a particular demand.

The AP-120B is a completely synchronous device with a cycle time of 167 nanoseconds. Every instruction takes exactly one cycle, although the results of an operation initiated in one instruction may not be available at the next instruction, but only one or more cycles later. The synchronous behaviour makes it very easy to determine the execution time of sections of code, provided no communication with the host is taking place. For example, a loop which is 10 instructions long, with no branching, will require exactly 1.67 microseconds for each time around the loop. A true benchmark, must include the communication overheads, however, unless the complete program and all data reside in the AP and the execution time in the AP is long with respect to its startup time (a few milliseconds).

The AP-120B is a horizontally microcoded machine. By this, we mean that the instruction set is based on relatively primitive machine operations, such as register transfers along specific internal buses, and that an instruction contains a number of different fields, each field controlling one of a number of possible parallel operations. The instruction word is 64-bits wide, and up to 10 operations can occur in parallel. In practice, it would seem impossible to write code which has more than 5 or 6 operations occuring simultaneously and commonly there are 3. There are no macroinstructions, as defined in our discussion of microprogramming, and the AP-120B's microinstructions are the machine instructions. A microassembler (APAL) is available for symbolic coding of the various parallel microoperations.

The AP-120B has separate program and data memories. The data word is a 38-bit floating point word with a 10 bit exponent and a 28 bit 2's complement mantissa. This leads to approximately 8 decimal digits of precision over a wide range ($\sim 10^{\pm 150}$). Although a 64-bit machine has been announced by CSP, Inc., 38 bits is the best precision available in any currently available array processor. The FPS precision has to be considered minimal or inadequate for many scientific calculations. When it is adequate, another problem arises. In normal operation, host floating point formats and AP-120B floating point formats are converted "on-the-fly" by hardware in the interface. This means that for most hosts the 38 bits will be chopped to single precision 32 bits on passing data from the AP-120B to the host. This might be all right for some calculations in which only the final answer is passed to the host, but it is a problem with many calculations. A group at Cornell, where an AP-190L is attached to an IBM 370/168, requested and received from FPS an interface which converts 38 bits to double precision 64 bits. Whether this option is or will be available for hosts other than the 370/168 is not clear. Depending on the application, it might be possible to preserve the 38 bits by user written software, without severely slowing a calculation down.

A block diagram of the AP-120B indicating its data paths is shown in Figure 6. Neither the control unit nor the path from program memory to the control unit is shown. The architecture includes a floating point adder and a floating point multiplier. These are both pipelined units. The pipelines are reasonably short which is advantageous for non-vector operations, since the pipelines can be filled or drained reasonably quickly. The adder has 2 segments and the multiplier 3 segments. These can be compared with the corresponding 6 and 7 segments of the Cray-1. Thus, an independent add and multiply can be initiated every cycle (167 nanoseconds) but isolated additions and subtractions require 333 and 500 nanoseconds, respectively. The program memory is fast bipolar (TTL) memory and is currently relatively expensive ($3,500 per 1K words). By comparison, 1 Mbyte of MOS memory for a minicomputer can currently sell for less than $15,000. Addresses are limited to 12 bits by the architecture and the maximum program memory is 4K words

(4096 microinstructions). This is relatively small and is equivalent to perhaps 300 or 400 FORTRAN statements using the FORTRAN compiler that is available. On the other hand, it will seem very large if one tries to hand-code 4096 microinstructions. The main data memory is addressed by 16 bits and the normal maximum size is 64K words ( a new page select option is available which allows segmentation and an extension to one million words). The lack of a linear address space does not make more than 64K words usable in a simple convenient way, however. As shown in Figure 6, there is a data path from main data memory to program memory and it is possible for the user to write his own overlay routines to store programs in data memory (in 32 bit sections, right justified in two 38 bit words) and to transfer sections of code from main data memory to program memory as needed. To accomplish this is not a trivial programming job.

The AP-120B can include up to 64K words of fast (and therefore expensive) table memory. The standard option is 2.5K words of ROM programmed by FPS with constants for calculating sines, cosines, square roots, etc. It also includes constants like 1.0, 2.0, and $\pi$. While not offered as a standard option, it might be possible to obtain ROM programmed with the user's own set of constants, for polynomial evaluation of special functions, for example. A RAM option is also available. Some people have found RAM table memory very useful for storage of user constants, since they can be accessed in parallel with main data memory.

The internal registers (accumulators) include 2 sets of 32 registers - data pad X (DPX) and data pad Y (DPY). One each of DPX and DPY can be read and written in each instruction. Unfortunately, addressing of these registers is not totally straightforward and only 8 of each of DPX and DPY are available at a given time. Most user programs use only a few registers, however.

The S-PAD, which includes an ALU and sixteen 16-bit integer registers, is used for address arithmetic and integer operations. The AP-120B is deficient in its integer operations. The S-PAD ALU has no provision for integer multiplication or division. These operations are normally done, after conversion to floating point, in the floating point units.

The address registers MA (16 bits), TMA (16 bits) and DPA (5 bits) are used to address main data memory, table memory and the data pads, respectively. A 3 bit offset is added to DPA for each of DPX and DPY.

It is possible to add peripherals (for example, disks) directly to the AP-120B. The IOP is a hard-wired interface and the PIOP is a programmable (intelligent) interface. In applications that require large amounts of data transfer, it might be worthwhile attaching a disk directly to the AP-120B, in order to eliminate host overhead. These options are relatively new and no experience in their use has been passed on to the author. It seems that this option is not yet well supported by FPS.

As illustrated by Figure 6, the data paths of a machine like the AP-120B are an exercise in the theory of connectivity. For example, one operand (M1) of the multiplier can come from one of DPX, DPY, table memory, or the output of the multiplier. The other operand (M2) can come from one of DPX, DPY, main data memory, or the output of the adder. Bottlenecks do occur, as not all desirable direct transfers are possible, but the multiplicity of data paths is certainly one of the AP-120B's better features. Since the outputs of the adder and multiplier connect to the inputs of the adder and multiplier, vector operations can be chained.

Apart from the data pads, accesses to memory in the AP-120B require care in programming since initiation of memory references can not, in general, occur every instruction (cycle) and the result of a memory read will only be available 2 or 3 cycles after its initiation, depending on the type of main data memory used. The standard main data memory has a cycle time of 333 nanoseconds. With this memory a main data memory reference can be initiated only every other instruction. A memory reference is initated by changing the main data memory address register (MA). If the main data memory input register (MDI) is altered in the same instruction, the memory reference is interpreted as a write rather than a read. The result of a memory read is only available in the main data register (MD), ready for use, three instructions beyond that in which the reference is initiated. The main data memory is two-way interleaved using 16 banks of 4K words, where the three most significant bits and the least significant bit of the address determines the bank. If references refer to the same bank, then they must be separated by two intervening instructions rather than the normal one intervening instruction. For example, memory references to sequential addresses can occur every two cycles, but those to addresses spaced by two can occur only every three cycles. If the faster 167 nanosecond main data memory is purchased, then memory references can be initiated every instruction and the result of a read is available for use by an instruction two after that which initiated the read. Programs are not always transferable between units with different speeds of memory. Table memory uses a fast 167 nanosecond bipolar memory for which a read can be initiated every instruction and data is available for use in the second instruction following.

The software for the AP-120B is not ideal but is probably the most extensive available for any array processor. It includes APEX, an executive routine, written in FORTRAN and host assembly language, which resides in the host and interfaces to the host operating system. If standard FORTRAN calls are made to a library of array processor routines from a standard application FORTRAN program, then APEX will automatically handle the loading of library routines into the AP-120B. It similarly will handle FORTRAN calls made to library routines to transfer data in and out of the array processor, initialize the array processor, etc. A considerable library of mathematical routines is available, including Householder diagonalization of matrices. It is unlikely, however, that the programs of computational chemistry can get by with only the standard subroutine library. To add one's own subroutines to the library requires coding in APAL, the array processor assembly language. The assembler is a cross-assembler, written in FORTRAN, which normally runs on the host to produce a relocatable output object deck. Object decks are then run through APLINK, again a FORTRAN program which runs on the host, to produce load modules which are actually FORTRAN subroutines containing a call to APEX and the array processor microcode in DATA statements. These FORTRAN routines are then included with one's own FORTRAN application program. A call to one of these library routines results in a call to APEX, which loads the microcode into the array processor if it is not already there. If many small library routines are called, the overhead in the host can be very high. To alleviate this problem, a vector function chainer (VFC) is available which allows one to combine a number of sequential calls to library routines into a single call.

In addition to the above software, a simulator (APSIM), a debugger (APDBUG), and various hardware diagnostic routines are available. The simulator is written in FORTRAN and runs on the host. Using it, one can simulate array processor programs and their execution times without having an array processor. Most programs are debugged on the simulator prior to ever attempting to execute them on the AP-120B.

A FORTRAN compiler has been announced by FPS and is apparently now available, although it is impossible to know at this point how efficient (or inefficient) it will be. Writing a compiler for an instruction set like that of the AP-120B is a research problem and is not a simple task. The FPS compiler derives from one written at Cornell University and it probably will generate code comparable to that of the Cornell compiler. Both compilers are written in FORTRAN and are large programs. The Cornell compiler requires over 700K bytes of memory. The size of the FPS compiler is not known but it will certainly not fit in 256K bytes and probably not in 512K bytes. It is thus not usable, for example, on a PDP-11/23 host. Because every AP-120B instruction requires exactly 167 nanoseconds, the efficiency of the compiled code is a simple direct function of the number of AP-120B instructions generated. The expansion factor, i.e., number of AP-120B instructions per FORTRAN statement, seems to be between five and twenty with ten as an average. This compiler generated code is described as being 5-10 times larger than hand-coded APAL instructions. Using a FORTRAN compiler thus limits one to something in the vicinity of 400 FORTRAN statements and an execution time 5-10 times longer than that obtainable using APAL. Since it is possible to initiate a floating point addition and a floating point multiplication every cycle (1/6 microseconds), the AP-120B has a theoretical upper limit of 12 MFLOPS (million floating point operations per second). If operands come from the data pads, it is possible to code a dot product routine with an inner loop only one instruction long and thus attain this upper limit of 12 MFLOPS.

If, however, operands come from main data memory which will almost always be the case, then, since a memory reference can only be initiated every other cycle, the inner loop is necessarily four instructions long (2 operands) and the dot product runs at 3 MFLOPS. The code generated by the Cornell compiler for a dot product, which may be taken to be indicative of the state of the art in automatic code generation for the AP-120B, has an inner loop which is 34 instructions long. Thus, a dot product using code generated by the FORTRAN compiler runs at only 0.35 MFLOPS. For comparison, the dot product on a VAX-11/780 with floating point accelerator runs at about 0.26 MFLOPS if operands come from the Cache. In addition, the AP-120B numbers do not include any possible host overhead. The above numbers problably show the AP-120B in its worst possible light but they do indicate the loss of efficiency with compiled rather than hand-generated code.

To illustrate, if nothing else, the complexity of coding in APAL, the code for a dot product $C = \sum_{i=0}^{N-1} A_i B_i$ is given below:

```
     A  $EQU 0 "BASE ADDRESS OF A
     I  $EQU 1 "INCREMENT FOR A
     B  $EQU 2 "BASE ADDRESS OF B
     J  $EQU 3 "INCREMENT FOR B
     C  $EQU 4 "ADDRESS OF C
     N  $EQU 5 "VECTOR LENGTH
DOTPR:    MOV A,A; SETMA "FETCH A(0)
     NOP  "WAIT FOR MEMORY
     MOV B,B; SETMA "FETCH B(0)
     DPX←MD;  "SAVE A(0)
       INC N   "KEEP COUNT RIGHT
     ADD I,A; SETMA "FETCH A(1)
       FADD ZERO,ZERO    "CLEAR SUM
LOOP:     FMUL DPX,MD;    "DO A(I)*B(I)
       FADD;   "PUSH ADDER
       DEC N   "SEE IF DONE
     BEQ DONE; "BRANCH IF DONE
       FMUL;   "PUSH MULTIPLIER
       ADD J,B; SETMA    "FETCH B(I+1)
     DPX←MD;  "SAVE A(I+1)
       FMUL    "PUSH MULTIPLIER
     FADD FM,FA;    "ADD (A(I)*B(I)) TO SUM
       ADD I,A; SETMA;   "FETCH A(I+2)
       BR LOOP "BRANCH BACK
DONE:     MI←FA;MOV C,C; SETMA;   "STORE RESULT IN C
       RETURN
     $END
```

Semicolons here separate parallel operations in the same instruction, i.e. indentation continues the same instruction. A,I,B,J,C, and N are S-Pad, 16-bit integer registers. A memory read here is initiated by setting the main data memory register (SETMA) with the result of an S-Pad operation performed in the same instruction. Thus, MOV A,A; SETMA moves the contents of register A back into register A only to present an address for the SETMA operation. The initiation of a memory write, such as in the last instruction of the program, is identical except that the main data memory input register (MI) must be loaded (in this case with the output of the adder (FA)) in the same instruction. In the other memory references, such as ADD I,A; SETMA, register A is incremented by the contents of register I and the result is available in the same instruction for SETMA. Since memory references can occur only every other cycle, a "no-operation" (NOP) is necessary at instruction two. The result of this first memory read, is available in the main data memory registrer (MD) only at the fourth instruction (3 instructions after the reference is initiated), where it is temporarily stored in one of the data pad registers (DPX MD;). Operands are entered into either the adder pipeline or the multiplier pipeline by the instructions FADD and FMUL with two operands. These initiate the first segment of each pipeline. Operands will not flow through the pipeline on their own, however, and must be pushed through by executing subsequent FADD or FMUL instructions. Without arguments, these instruction do not enter new operands into the pipeline but simply push existing operands on to the next segment of the pipeline. In this example, the three segment multiplier requires two pushes and the two segment adder only one. The outputs of the adder and multiplier (FA and FM) remain in these locations until replaced by the result of new additions or multiplications.

Although not extreme in this example, careful inspection will show the way parallel execution of more than one operation in an instruction enables one to "wrap" code around in a loop once the loop is properly initialized. In this example, the highest degree of parallelism is essentially only three. For example, the second to the last instruction includes an addition, the initiation of a memory reference, and a branch. This loop is an example of a memory limited loop. Since it includes two references to main data memory, the loop must be at least four instructions long. As indicated previously, if operands come from separate data pads, DPX and DPY, the loop, after appropriate intialization, could be reduced to the single instruction,

```
LOOP:  FMUL DPX, DPY;  INCDPA;  FADD FM, FA;  DEC N;  BGT LOOP
```

Here, INCDPA increments the data pad address register.

As the example illustrates, programming the AP-120B is not for the faint of heart. Most chemists are not accomplished assembly language programmers and microcoding the AP-120B has been described as between 2 and 10 times more difficult than normal assembly language progamming. If an AP-120B is used for chemical computations, a choice must be made between the inefficient code produced by a FORTRAN compiler and the very large investment in time and effort required to generate APAL programs.

The present price of a representative configuration (without RAM table memory) of an AP-120B for attachment to a VAX-11/780 is shown in Table 1.

Table 1.
Current Prices[1] for a Representative AP-120B Configuration

| Part No. | Description | Price |
|---|---|---|
| AP-120/664 | AP-120B Configuration[2] with 64K Words of 38-bit Fast Main Data Memory (DMF32); 1024 Words of Program Source Memory; 2.5K Words of ROM Table Memory; and Standard Math Library. | 73,360 |
| AP-PS1024 | 1024 Words of Program Source Memory | 3,760 |
| AP-PS2048 | 2048 Words of Program Source Memory | 7,145 |
| AP-DE03-I | Computer Interface; DEC VAX 11/780 (Unibus) | 6,000 |
| AP-DE06-S | Software Interface; DEC VAX 11/780 (Unibus) | 6,000 |
| AP-PDS-DE04 | Program Development software[3] | 3,000 |
| AP-Fort79.1 | FORTRAN compiler for array-processor | 8,500 |
|  | TOTAL | $107,765 |

NOTES:

1 As of June 1, 1979.
2 Contains APEX (operating system driver), AP-TEST (diagnostic software for the AP-120B), system installation and acceptance.
3 Contains APAL (array-processor assembly language), APLINK (linker for code produced by assembler and FORTRAN), APSIM (software package to simulate array-processor actions on the host), APDBUG (a debugger for AP software), and VFC (the vector function chainer).

## 4.2 CSP Inc., Burlington, MA 01803

The main competition and perhaps the only competition, with respect to the calculations of computational chemistry, for FPS comes from CSPI (originally Computer Systems Products, Inc.). CSPI has announced a 64 bit array processor, the MAP-6400, which is expected to be available early in 1980. Little explicit information is yet available on the MAP-6400 so the MAP-200, which it will most closely resemble, will be described first. CSPI also produces other related 32-bit machines - the MAP-100 and MAP-300.

The MAP-200 is a 32 bit floating point array processor which uses the IBM single precision floating point format (about 6 decimal digits). Unlike the AP-120B it has independent asynchronous processors which each execute separate processes. A block diagram of it is shown in Figure 7. Each processor is attached to three independent buses and associated memories. A number of I/O processors (scrolls) can be attached to it. Apart from the I/O processors, the unit contains four asynchronous processors: the central signal processing unit (CSPU), which is the master of the other three; a host interface scroll (HIS), which is responsible for transfering data and programs from the host to one of the three memories; an arithmetic processing unit (APU), which contains a small number of registers and a floating point multiplier and adder; and an arithmetic processor scroll (APS), which is responsible for address arithmetic and fetching operands from memory for the APU and storing in memory the results of APU calculations. Both the APU and APS are contained in the arithmetic processor of Figure 7. Apart from the CSPU, each of the processors has its own small program memory -- 256 16-bit words for the APU, 128 25-bit words for the APS, and 64 32-bit words for the HIS. The CSPU's program is stored in one of the three main memories. Thus, there are four (apart from attached I/O devices) independent instruction streams. The CSPU is a reasonably fast (125 nanosecond cycle time) processor with a general purpose integer-only instruction set. It is responsible for loading the small local program memories of the APU, APS, and HIS from one of the three main memories and, in general, it controls all of the other array processor resources upon command from the host. Communication between processors occurs via hardware flags and an elaborate interrupt mechanism. The APU is data driven; it has a queue of operands, which is filled by the APS. If the queue is empty, the APU idles. This independent asynchronous generation of operands and operations probably works fine for simple vector operations on long vectors but computation could be overwhelmed by communication problems if the code includes much branching (the GO TO and IF statements of FORTRAN). A normal philosophy of multiprocessor programming is that, because one has independent instruction streams, one can and should maximize control (branching) within a process and simultaneously minimize communication between processes. The special CSPI architectural relationship between the APU and APS does not allow this. Control requires expensive communication. The MAP-200 thus, in some ways, resembles a SIMD machine where one must minimize control at all costs. In the author's opinion it has most of the disadvantages of any MIMD machine but fewer of the advantages.

The instruction set of the MAP-200 is vertical rather than horizontal and, apart from the multiprocessor aspects, programming it is similar to that of normal assembly language programming. One would thus expect it to be easier to program than the AP-120B. Users report, however, that its asynchronous nature can make it very difficult to program. Between the FPS and CSPI machines one has the trade-off between difficult horizontal microcode but simple synchronous behavior (FPS) and simple vertical instruction set but difficult asynchronous behavior (CSPI). The software available includes a math library (insufficient for many computational chemistry applications), an assembler and simulator, which are written in FORTRAN and run on the host, and appropriate interfaces to the host operating system. A FORTRAN compiler is planned but has not yet been announced. The CSPI concept of a function list is analogous to FPS's vector function chainer, but is apparently easier to use.

The MAP-6400 will merge two of the 32-bit data buses and their associated memories into a single 64-bit data bus and a single data memory. The remaining 32-bit bus and memory will be used exclusively for programs. A system with 32K of 64-bit data memory and 16K of 32-bit program memory will apparently cost in the vicinity of $89,000. This configuration includes slow (500 nanoseconds) memory but faster and/or larger memories will be available. After other cost are included, the price is likely to be similar to that quoted above for the AP-120B. The basic multiplication time of the MAP-6400 is slightly under 1 microsecond. Addition takes considerably less time. If an addition is requested immediately after a multiplication, the two operations will be executed in parallel, in the time of the multiplication. Thus, one addition and one multiplication require one microsecond (2 MFLOPS) and CSPI quotes 1 second for a 100 x 100 matrix multiplication. They quote an unknown configuration of the VAX-11/780 to require 12 seconds for the same matrix multiplication. The MAP-6400 is quoted as performing a 50 x 50 matrix inverse in 227 milliseconds. The corresponding AP-120B time (333 nanosecond memory) is 202 milliseconds, not including any possible overhead. These numbers make the MAP-6400 appear reasonably attractive, at least for straight-forward matrix operations, particularily since these times are for 64-bit rather than 38- bit arithmetic. One problem with the MAP-6400's 64-bit floating point word is that user software will be needed to convert to other than IBM's format on passing to and from a host. The MAP-6400 has a theoretical upper limit of 3 MFLOPS (assuming two additions can be hidden behind a multiplication rather than just one, as described above). The AP-120B has a theoretical upper limit of 12 MFLOPS but commonly runs at 2-4 MFLOPS for memory access limited loops, although there is at least one example of a chemically significant inner loop which runs at 10 MFLOPS, as discussed later.

## 4.3 Datawest Corporation, Scottsdale AZ 85260

The Datawest Model 480 array processor is an interesting but not inexpensive product (about $500,000 for a basic configuration) which attaches to UNIVAC mainframes. It contains four multipliers and eight adders and is theoretically capable of 120 MFLOPS. The floating point format (36 bits, equivalent to the UNIVAC format) uses only 8 bits for an exponent and the 480 thus has the identical precision of the AP-120B, without the problem of chopping to 32 bits upon transfer of data to the host.

It is a multiprocessor with four processors, called slices, and four instruction streams. Unlike other multiprocessors, it is synchronized. The four processors can execute different instructions but there is only one program counter and each processor must execute exactly the same number of instructions. Each processor is horizontally microcoded with a 72-bit microinstruction. Alternatively, since there is only the single program counter, the four simultaneous instructions can be described as a single instruction which is 4 x 72 = 288 bits wide. Only one of the processors is capable of executing branch or jump instructions and the other three remain idle during a branch. Each processor has a number of registers, a floating point multiplier, 2 floating point adders, and an integer ALU. The floating point multiplier and adders are pipelined with 4 segments each. The cycle time is 100 nanoseconds so that 400 nanoseconds is required for an isolated addition or multiplication, but only 100 nanoseconds if the pipelines are kept full. The program memory size is 8K 36-bit words expandable to 64K. This is equivalent to 1024 288-bit microinstructions. Every 100 nanosecond cycle a 72-bit microinstruction is executed by each of four processors. The data memory size is 64K 36-bit words expandable to 1024K. Each processor can access data simultaneously. The bandwidth (data transfer rate) for communications with the host can be made very high (40M words per second) if four interface ports are used.

It would appear that as an array processor, the Datawest 480 has promise of being a very high performance machine and those installations using a UNIVAC machine would do well to consider it carefully. The software available includes an executive with multiuser capability, microassembler, simulator, etc. Again, a major investment in effort would be required to develop software, since high-level languages are not available.

## 4.4 Other Products

There are few remaining products that in the author's opinion are of interest for computational chemistry applications. IBM produces an array processor, the 3838, but it is considerably more expensive than the FPS and CSPI products and is limited to 32-bit single precision arithmetic. Data General produces an array processor which is integrated into an ECLIPSE S/130 minicomputer. Its price is quite reasonable but it is also limited to 32-bit single precision arithmetic. In addition, its microcode is in ROM and not programmable. Only a few standard vector operations, mostly oriented toward FFT's, are available with the S/130. This array processor, however, has 64-bit buses (used for 32 bit real and 32 bit imaginary parts) and could conceivably be turned into an attractive 64 bit machine by eliminating complex arithmetic, replacing the 32-bit multiplier and adder by their 64-bit counterparts, and using RAM for the microcode. It is unfortunate that almost all array processors were developed for signal processing applications rather than high precision scientific computation. It would be very desirable to make better known to manufacturers the needs of the computational scientific community.

## 5.0 User's Experience

A number of chemists are thinking of getting array processors or perhaps even have written proposals for the funding of one. So far, however, the experience of the computational chemistry community with array processors is still rather limited. Here we try to summarize the limited experience of researchers who have used array processors for problems related to those of computational chemists. We have been unable to find anyone using a CSPI array processor for other than signal processing applications, so remarks will need to be confined almost exclusively to user's experience with the Floating Point Systems' AP-120B. Until chemists or physicists have experience with the MAP-6400 it will be a difficult product to evaluate.

The first array processor in chemistry (serial #2) was obtained by the group of Professor Kent Wilson at the University of California, San Diego. As one of the first FPS products, the hardware was unfortunately plagued by difficulties. Eventually, it was replaced by a new model. Since then, it has been quite reliable. Their AP-120B has 1.5K of program memory and 64K of data memory, although they managed with much less data memory for some time. It is attached to a microcoded California Data computer emulating the instruction set of a PDP-11/40, although it will be transferred to a VAX-11/780 in the near future. This group has used the AP-120B mainly for molecular dynamics simulation of molecular motion and chemical reaction. All of their programs have been written in APAL and their complete molecular dynamics programs fits into their minimal 1.5K program memory. Without great data handling problems and with the total program residing in the array processor, this group has not experienced problems with host overhead and data transfers. It is difficult to quote a megaflop rate for their code, since they evaluate the potential by a table lookup. They quote their code as containing 2-3 operations per instruction. After some years experience, this group appear to be using an AP-120B very successfully in molecular dynamics calculations.

Considerable effort has gone into array processors at Cornell University. An AP-190L is attached to the university's IBM 370/168. Two thirds of it was purchased by a group of physicists, including Professors K.G. Wilson, G. Chester and others. The remaining third was purchased by the computer center for general use on campus. A group of individuals, headed by Dr. Alec Grimison, in the Office of Computer Services, has been responsible for writing approximately 2 man-years worth of software which make their system usable by the average FORTRAN programmer. This software includes APEMAN, an array processor execution manager which interfaces to the VM/370 Conversational Monitor System of their 370/168 and schedules user's jobs for execution on the array processor, as well as the FORTRAN compiler mentioned earlier. Both pieces of software are available to academic institutions - $3500 for APEMAN and $5000 for the FORTRAN compiler. In spite of the difficulties of producing efficient code from a FORTRAN compiler, it is thought that compiled jobs run on the AP-120B about as fast as they run on the 370/168. The main reason for having an array processor and the prime reason why considerable effort has been spent developing software for array processors is the cost to the user - $1476 per hour for the 370/168 versus $40 per hour for the AP-190L. Their configuration includes 4K of program memory and 96K of main data memory, although only 64K are accessible to the FORTRAN programmer (the compiler does not understand the segmentation mechanism which allows one to access more than 64K). A small amount of RAM table memory is available but is not used by the compiler. It is a bit unclear what kind of jobs are being run by the general user. Since there is a limit of approximately 400 FORTRAN statements internal to the array processor and an imposed time limit of 15 minutes, it can be assumed that many are small jobs, mainly run on the AP-190L for its cost - effectiveness. Longer jobs can be run provided thy are checkpointed and go to the end of the queue every 15 minutes. Very large tasks, which restrict themselves to a sequence of calls to subroutines of fewer than 400 FORTRAN statements, can thus be run in a sequence of 15 minute time slices.

The scientists who own 2/3 of this machine use it mainly for the Monte Carlo simulations of statistical mechanics. Monte Carlo programs have small code and, unless configurations are kept for some later use, very few data handling problems. They are thus probably ideally suited for an array processor such as the Floating Point Systems' machine. Those with experience in these codes claim that, at least for their applications, the AP-120B (or AP-190L) is as much a very fast scalar processor, as it is an array processor. That is, since the pipelines are quite shallow, one need not have the normal very long vector operation to obtain high throughput. The degrees of parallelism of this machine are as important as the pipelines. A recent paper[14] outlines some of the Cornell group's benchmarks and experience with their AP-190L.

A second group at Cornell headed by Professor H. Scheraga, of the Chemistry Department, in collaboration with Professor C. Pottle of the Electrical Engineering Department, have attached an AP-120B to a Prime 350 minicomputer. Their configuration includes 32K words of main data memory, 2.5K words of program memory and 1K words of RAM table memory. The AP-120B will initially be used for molecular mechanics calculations on peptides, i.e., to find the minimum energy conformation of small enzymes. The computationally intensive part of their calculation, i.e., the calculation of the energy and its gradient, has been written in APAL and reside wholly in the array processor, while the outer sections of code, including the non-linear minimization, are run on the host. They have only recently gotten their array processor running, but a few explicit results are already available. Initially, they have minimized the energy with respect to 154 dihedral angles of Bovine Pancreatic Trypsin Inhibitor (886 atoms) using a coulomb interaction of point charges plus a 6-12 Lennard-Jones type potential for all non-bonded interactions. A large amount of thoughtful effort has gone into the APAL programming of the compute-bound inner loop of this calculation. The very tight inner loop runs at 10.3 MFLOPS! Coding such an inner loop is not for the amateur. The BPTI calculation required 27 hours of AP-120B time plus 3.4 hours of host time. They estimate this run would have cost $50,000 if run on Cornell University's 370/168. This group has used RAM table memory for storage of Lennard-Jones parameters and for table lookup of $(R^2)^{-1/2}$ and have found it very valuable, since it can be accessed in parallel with main data memory and the data pad registers. The Scheraga group is likely to use an array processor very successfully. Some of the reasons for this have to include the size of the group, which is fairly large by current academic standards. It includes a senior research associate, half a dozen postdocs, two or three full-time computer support staff and several graduate students. In addition, the active collaboration of an Electrical Engineering faculty member with expertise in computer science and parallel computations, as well as the wide experience of the Cornell campus with array processors must be taken into account. Also, the problems, being solved, including Monte Carlo calculations of water and aqueous solutions, are among those best suited for the AP-120B. Their experience does not completely answer the question of whether a small research group with little computer science expertise can successfully operate an AP-120B. The chief limitation of their Prime 350, AP-120B configuration is the minicomputer (only 196K bytes of memory, 6 Mbytes of disk storage and no tape drives) rather than the array processor.

The Center for Plasma Physics and Fusion Engineering at UCLA have a special configuration of the AP-120B, under the direction of Professor J.M. Dawson. Their system was designed and implemented by CHI Systems of Santa Barbara. Glen Culler of CHI Systems is responsible for the initial design that lead to the Floating Point Systems' product. The UCLA system is used mainly for plasma simulations. These calculations are similar to the molecular dynamics calculations of chemistry except that the pure coulomb force allows an electric field to define the force on a particle, eliminating the necessity of summing discrete interactions. As many as a million particles can be treated. The large number of particle coordinates and electric field components require external mass storage and rapid data transfers. The UCLA CHI system has four disks directly attached (through intelligent I/O processors) to the AP-120B, to facilitate this movement of data. The total bandwidth is about $10^6$ 38-bit words per second. The host is a specially designed 16-bit fixed point processor with a 167 nanosecond cycle time. An integer multiplication requires only 2 cycles. As well as FPS software (APAL, math library), there is special purpose software associated with the host. One piece of this software is a Math System Language which interprets high-level instructions. Programs are primarily written in the Math System Language except that critical portions of production type codes are coded in APAL. The system can be used interactively by a number of simultaneous users. The time-shared operating system uses time slices of only a single second duration and the user with long compute-bound jobs has to write his programs such that they execute in very short steps. In normal operating systems, time slicing is invisible to the user. The AP-120B, however, was not designed for time sharing and has no interrupt mechanism etc. While the UCLA system may work well for the plasma simulation problem, it is the author's opinion that for the problems of computational chemistry the AP-120B is better dedicated to the batch execution of long computationally intensive jobs. One might, of course, still time share on the host. For the plasma simulation problem, the UCLA CHI system runs about three to four times faster than an IBM 360/91 and is quoted to be two orders of magnitude more cost-effective. Like most estimates of cost-effectiveness this does not, of course, include the man-hour cost of software generation.

The group of Professor McTague in the Chemistry Department of UCLA are planning to use the UCLA CHI system for molecular dynamics simulation of argon on graphite (2000 atoms in two dimensions) Perhaps, something on the order of a man-year (1 graduate student!) has been invested in these calculations and the program is apparently nearing completion. The UCLA AP-120B has only 512 words of program memory but with careful assembly language (APAL) programming the basic molecular dynamics routine will apparently fit. The lack of hardware divide presents problems as usual, so that the forces will be interpolated from a table, as a number of others have also seen fit to do, instead of evaluating them explicitly. Table evaluation of forces (or potentials) is probably an appropriate technique to use for the AP-120B, because of the difficult recoding problem encountered when the form of an explicit force (or potential) is changed.

All of those doing Monte Carlo, molecular dynamics, or molecular mechanics calculations with the AP-120/B have chosen, willingly or not, to code their critical sections of code in APAL. With effort, they would all appear to have, or soon have, very cost-effective approaches to a subset of the main computational problems of chemistry. No one has complained bitterly that 38 bits of accuracy is inadequate. Part of this must be the choice of problems, since even more than 64 bits of precision appears to be required for the very long classical trajectories of some chemical reactions. The users who have chosen their problems carefully, appear to be genuinely satisfied. This is not always the case, as illustrated by a particular graphics application on an AP-120B attached to a PDP-11/70. Even with hand-coding this application (assumed here to be translation and rotation of a graphics image) ran no faster with the AP-120B than without it. This implies that if the AP-120B was attached to a VAX-11/780, which was the original intention, the application would run about two times slower with the AP-120B than it would on the VAX alone (the VAX-11/780 is about twice the speed of a PDP-11/70). Apparently, the overhead in the operating system (UNIX) is so high, and so few calculations performed in the AP-120B per datum fetched from the host, that computation is swamped by data communication problems.

## 6.0 Discussion

Many questions still need to be asked about the proper role of array processors in computational chemistry. A few of these are indicated in the appendix in association with the meeting held at NRCC. Industry has probably been quite capable of producing the "ideal" machine for computational chemistry, but unfortunately it has not. Does the scientific community, particularily the academic community, present enough of a market to influence the production of devices which satisfy our needs? What are the needs of the computational chemistry community? Many more questions can be asked than answered. If our needs are to be met by manufacturers and if we are to reap the benefits of the new VLSI technology, it may be necessary to align ourselves with others with similar requirements. The electric power industry, through the Electric Power Research Institute, Palo Alto, has a program in operation which is designed to thoroughly evaluate array processors, including in-house experimentation, in order to decide, as best as possible, what is the optimum architecture for their needs, and exert pressure on manufacturers to satisfy these needs. The NRCC is many ways is a very visible organization, representing a great number of chemists. Should it, and can it, expand its horizon to include close contacts with hardware manufacturers, the semiconductor industry, etc, in order to lobby on behalf of computational chemists? Should it even undertake a step in the direction of architectural research? Computational chemistry is now a resonably mature subject with at least some well defined needs. Can we assume that in the normal course of events, industry will produce those devices with which we will be satisfied? Chemists have led the way in designing and building very sophisticated molecular beam machines, ion cyclotron resonance machines, etc. Should computational chemists consider only software and leave the hardware to others?

The question remains as to how chemists should compute in the next 2 years, 5 years, or 10 years - with a supercomputer, a minicomputer, an array processor, or some future architecture? The answer is probably, with all of these. Array processors will certainly become more powerful. Floating Point Systems, for example, is expected to announce a new 64 bit array processor within the next year. Other array processors can be expected to appear. Minicomputers, with an attached array processor when appropriate, will probably remain a very cost-effective means of computation, at least until the effects of VLSI start having their impact on high performance computation, perhaps in five years.

To return to the more mundane present it is still pertinent to ask where and when the current models of array processors could or should be used for chemical computations. Until more chemically related experience is available with the CSPI MAP-6400, it must remain an uknown product. Thus, until other products appear on the market place, the only array processor with a precision beyond 32 bits which seems to merit very serious consideration by individuals or chemistry departments is the FPS AP-120B. With this or any other array processor there remains the question of true cost-effectiveness, given the sometimes huge investment in software that must be made. This investment can be minimized if a FORTRAN compiler is used, but the AP-120B may then not provide the performance which, in the long run, makes using it advantageous relative to the general purpose, well supported super-minicomputers (which will continue to drop in price as the market expands and newer technology is used in building them). Even with a FORTRAN compiler, the AP-120B will almost surely require more support facilities than an off-the-shelf minicomputer.

The problems of Monte Carlo or molecular dynamics simulation, the classical trajectories of chemical kinetics, and molecular mechanics calculations seem to be well suited to the use of an array processor. They all require small code which would fit into the small program memory of an AP-120B (perhaps not, if a FORTRAN compiler is used), are compute bound, and can be formulated to minimize communication with the host. They also can require vast amounts of computational time which would be prohibitively expensive on a mainframe like the IBM 370/168, if the user was charged "real" dollars. Provided the precision problem is carefully weighed and sufficient manpower is available to code at least critical inner loops in microcode, the AP-120B should be seriously considered for the above problems.

Perhaps the major user of computational resources in chemistry is the quantum chemist. Can electronic structure calculations be performed on an array processor? For example, can Gaussian 70, Gaussian 76, etc. be executed on the AP-120B and, if so, can they be executed efficiently? The answer, except in certain cases, is probably no. The standard _ab initio_ molecular orbital calculation, as an example, consists of an integral evaluation step and an SCF step. Depending on the program, size of molecule, basis set, etc., these two steps take the same time to within a small factor. The bottleneck of the SCF step is the simple multiplication of an integral by appropriate density matrix elements and adding the result to appropriate Fock matrix elements. This is far too few floating point operations per fetch of an integral from the host (or disk) to run effectively on the AP-120B. On the order of a few hundred floating point operations per fetch of a datum are probably necessary for the AP-120B to perform well. A ballpark number for transfering data on the Unibus, when blocks of 1K words are used and the operating system is UNIX, is 16 microseconds per 16 bit word. Dr. George Purvis of Battelle Laboratories has estimated that with the above transfer time a 146 x 146 matrix multiplication (292 floating point operations per fetch) would be speeded up by about a factor of 10 over the VAX-11/780 time, if the data came from a disk attached to the VAX. This optimistic number will decrease rapidly, however, as the number of floating point operations, per datum fetched, falls. It cannot be expected that the SCF step of an _ab initio_ calculation will be speeded up over that of a good minicomputer. The array processor will be mainly idle waiting for data.

The integral evaluation step is not easily vectorized because of the large number of different types of integrals. Nevertheless, the large scalar speed of the AP-120B could be put to effective use here, but only if a significant fraction of the code could be stored in the array processor. It would be a hopeless pursuit to try and manipulate the very large integral code of Gaussian 70 into the AP-120B. The newer integral programs which use Rys polynomials (HONDO, for example) could possibly be run on an array processor, if the critical sections of code were isolated and if they could be made to fit into the 4K program memory of an AP-120B. Even if the integral evaluation time could be significantly reduced, the time for the SCF step would still remain and the reduction in execution time of the overall procedure would not be large. A number of groups are now evaluating analytical derivatives of the molecular orbital energy with respect to nuclear motion. In these calculations, where the SCF step constitutes a much smaller fraction of the total execution time, an array processor would be quite effective provided, again, that integrals (and derivatives of integrals) could be evaluted in the array processor.

Whether some of the many methods which include electron correlation could execute efficiently with the AP-120B is an open question. One area where an AP-120B might be useful is with semi-empirical calculations (CNDO, INDO, MINDO, etc.) on very big molecules. These calculations do not have as a bottleneck the problem of generating or manipulating vast numbers of two-electron integrals. For many of the problems that one would visualize solving with semi-empirical methods, however, the Fock matrix, density matrix, etc. would not fit into a 64K main data memory.

None of the immediately above discussion mentions the precision problem of the AP-120B. While some useful electronic structure calculations could be performed with only 38 bits, the general electronic structure calculation could not, particularily, if data words were chopped to 32 bits (6 decimal figures) by the interface. Perhaps, ways will be found to put an AP-120B to effective use in electronic structure calculations but this does not appear to be the most profitable direction in which to proceed.

The author is not familiar enough with the techniques and programs of crystallography to judge whether an array processor such as the AP-120B would be useful to crystallographers. Initial indications are that it would, but further exploration is required.

A few benchmark-type numbers have already been mentioned in the text above. It is clear that the AP-120B can be very fast. For some sparse matrix problems[16] it has beaten the CDC STAR-100, out-performed the CDC 6500 (dual 6400's) by 50 to 1, and even come close to the Cray-1 (on an algorithm which the Cray-1 handles very badly). The following numbers compare it with the VAX-11/780 for the standard Whetstone benchmark. This benchmark measures the number of instructions executed per second for a presumed average mixture of high-level instructions. These numbers were contributed by Floating Point Systems through Professor Kent Wilson.

<u>Standard Benchmark</u>

| Configuration | Whetstones |
|---|---|
| VAX-11/780 without DEC's floating point accelerator | 711 |
| VAX-11/780 with DEC's floating point accelerator | 1100 |
| VAX-11/780 with AP-120B and FPS FORTRAN compiler | 2580 |
| VAX-11/780 with AP-120B and hand-coded APAL | 14,000 |

These numbers illustrate the AP-120B's possible speed as well as the expected inefficiency of Floating Point Systems' compiler. A good description of Cornell's AP-190L attached to their IBM 370/168 has been published[14]. This paper includes a number of interesting benchmarks including benchmarks for Monte Carlo code (0.63 times the speed of a CDC 7600 and 1.5 times the speed of an IBM 370/168). Since the performance of the AP-120B is sensitive to the problem being solved, the host computer and its operating system, and the algorithm used and the method of coding it, there is still a need for more and better benchmarks for chemically significant problems.

Given that it appears an array processor can be appropriately applied to one's application, a few points still require discussion. Should it be attached to a university-wide computation facility, as in the Cornell example, or to a minicomputer dedicated to the use of a small group? It is the author's feeling that an array processor will find its best use when dedicated to a small number of specific problems requiring enormous amounts of computer time. Such problems are prevalent in computational chemistry. Provided sufficient manpower can be dedicated to maintaining and programming the facility, an array processor could extend, by perhaps an order of magnitude, the complexity of a certain number of chemically interesting questions which could be attacked by practical computations.

Attaching an array processor to a university wide resource will require a substantial commitment on the part of a central computer center to educating themselves in a new hardware device, maintaining and writing new system software, and down-playing their existing computational facilities. In Cornell's case, the Office of Computer Services has made a substantial commitment to support an array processor and the specific needs of a small group of physical scientists. Such a commitment is not likely to be generally available, nor is the required expertise. Another factor in favor of the success of the Cornell venture has been the very high cost of using their IBM 370/168. In many universities, at least small amounts of time will be available gratis, eliminating much of the demand for running short jobs on an array processor with its accompanying extra programming requirements and constraints.

The very large software problems still remain. Chemists have effectively abandoned programming in assembly language except perhaps for isolated situations where short critical routines are made as efficient as possbile. Even then, one normally has backup FORTRAN versions for portability reasons. Is it advisable for chemists to expend considerable effort programming a specific array processor, given a lifetime for the machine of, at most, 5 years? The obvious answer is to stick to FORTRAN but, as benchmarks have suggested, this may eliminate many of the speed advantages of an array processor. The best that can be suggested is that FORTRAN be used initially and that an effort be made to replace hopefully short critical sections of code with machine specific programming. It cannot be expected that someone will produce a super-efficient FORTRAN compiler in the near future. If a FORTRAN compiler is to be used with the AP-120B, it is certainly desirable to obtain the maximum (4K) of program memory. For those with a VAX-11/780, and data transfer problems, it may be desireable to wait until an AP-120B can be directly attached to the SBI rather than use the

Unibus interface. For those whose code will fit completely into an array processor and who do not have much data to be transferred, a relatively inexpensive configuration consisting , for example, of a PDP-11/23 microcomputer and an FPS-100 might be appropriate.

## 7.0 Array Processor Literature

Unlike the reasonably well defined communication channels of chemistry, the computer science and engineering literature often occurs in technical reports and other places difficult to access. The obvious first source is the manufacturers' brochures and manuals describing their products. To obtain other than the most superficial information, it is necessary to obtain the detailed manuals intended for actual users. The most relevant Floating Point Systems manuals are the "Processor Handbook" (a first introduction), the "Program Development Software Manual" (describing the use and syntax of APAL, APLINK, APSIM and APDBUG), the "Programmer's Reference Manual, Part I" (detailing the architecture and the way it is microprogrammed), the "Programmer's Reference Manual, Part II" (detailed list and description of each machine instruction), the "AP Math Library Manual, Volume 1" (introduction to the math library and its use) and the "AP Math Library Manual, Volume 2" (detailed description, except for actual code, of each math library routine and its execution time). In addition, manuals are available describing the IOP or PIOP peripheral processors and individual host interfaces, and each of the software modules, including the FORTRAN compiler.

Apart from a few uninformative promotional pamplets there is not yet any detailed information on CSPI's MAP-6400. A brief introduction to their current models is Document S-02, "An Introduction to the MAP Series Models 100, 200 and 300" Detailed information is contained in the "Macro Arithmetic Processor Systems (MAP-100/200/300) Programmer's Reference Manual" and "Simple Notation for Array Processing, Version II (Snap-II) Reference Manual." There are also a large number of other smaller manuals describing particular hardware or software aspects of the MAP series.

Datawest provides a reasonably informative small manual, "Datawest Series 480 Array Processor," to interested persons. The Eclipse Array processor is described in Data General's, "Eclipse AP/130 Array Processor Programmer's Reference," and "Array Processor Software (APS) User's Manual."

One of the best first introductions to array processors is a short article in Science by Arthur Robinson[17]. He gives the names and addresses of the principal array processor vendors.

Articles describing array processors and written by representatives of the various vendors periodically appear in various trade magazines[18-21]. These are almost always prompted by promotional considerations and cannot be taken as critical evaluations. Reference 18, however, does give the basic characteristics (word size, price, etc.) of fifteen different array processors. One article[22], by a well known architect, compares a few attached scientific processors but, unfortunately, does not contain a great deal of useful information. In the article and in a table, he compares the AP-120B with the Burroughs' BSP without noting the 12 million dollar price difference. One last trade magazine article[23], by a disinterested author, makes a reasonable comparison between the Floating Point Systems' AP-120B and CSPI's MAP-300. A short note by the present author[24], contained in the report of an NRCC workshop, does not contain much information but does have a short bibliography of papers on array processors, multiprocessors, and parallel algorithms. ( Note that Datawest no longer produces the MATP-400 array processor and the three technical memorandum from Computer Services, Cornell that are cited have been replaced by reference 25.)

As mentioned earlier, the Electric Power Research Institute (EPRI) is involved in evaluating array processors and parallel architectures, mainly for the solution of sparse linear equations. Three EPRI technical reports [16,26,27] will be of interest to those concerned with array processing. Unfortunately, some of these and other[28,29] interesting evaluations of array processors, including evaluations of the AP-120B, are for applications which are difficult to directly relate to those of computational chemistry.

To the disadvantage of a report like this, there is little yet published by those physicists and chemists who have explicit practical experience with an array processor. Professor Kent Wilson has described his system and related topics in two interesting papers[30,31], but he has not yet relayed, in print, a great deal of specific information on his group's use of the AP-120B. The best quantitative description on the use of the AP-120B in chemically related problems is a paper from Cornell[14], already noted. A good description of the UCLA CHI facility and its use in plasma simulation is given in reference 32. A paper describing experience with the Prime 350-AP120B system at Cornell has been submitted to the new Journal of Computational Chemistry.[33]

Floating Point Systems publishes the papers presented at its annual Users Group meeting, (1978[4] and 1979). These contain useful information for those having an AP-120B or those contemplating getting one. D. Bergmark's description of the Cornell FORTRAN compiler is contained in the 1978 report as is a preliminary description of Floating Point Systems' FORTRAN compiler. At the time of writing, the 1979 report was not yet available.

## 8.0 Role of the NRCC in Chemical Array Processing

By sponsoring this report and the meeting outlined in the appendix, the NRCC has already played a role in evaluating array processors. That is, it has attempted to inform the chemical community about array processors and it has intitiated a dialogue among interested persons on the subject of the present and possible future use of array processors in computational chemistry. What future role, if any, should it play with regard to array processors? As part of its informational function should it continue to keep chemists informed on specific commercial products, not only array processors but minicomputers, graphics systems, microcomputers, etc. Should it, attempt to meaningfully benchmark array processors? In addition to the large number of programs it now makes avaiable to chemists, should it either acquire externally, or attempt to develop in-house, software specific to particular machines such as the Floating Point Systems' AP-120B? Finally, should it attempt to lead in the use of array processors for chemical computation by acquiring one? These and many other questions concerning a specific role for the NRCC in the field of chemical computation with array processors could be asked. An attempt will be made to answer at least a few of these in the next section.

## 9.0 Recommendations

By the nature of this report it seems appropriate to make some specific recommendations. Many of these are implicit in the discussion above but it is best to make them explicit.

1: Those scientists with extreme computational requirements who are solving algorithms that do not make extensive use of mass storage and which have short, compute-bound code, should seriously consider an array processor as a cost-effective solution to their problems, provided the floating point precision is _known_ to be adequate for the application.

Comment: Anyone considering acquiring an array processor, however, should be fully cognizant of the software headaches he may be acquiring. The hidden cost of an array processor will be in software development.

2: The standard _ab initio_ molecular orbital calculations of quantum chemistry are not obvious candidates for any existing array processor.

Comment: With sufficient exploration there will probably be ways of putting a current array processor to use in electronic structure calculations, but an array processor is not a yet a tailor-made solution to the problems of quantum chemistry. Of the many methods for calculating correlated wavefunctions, some might use an array processor effectively, but extensive exploration of the applicability of a particular architecture to a particular application is necessary prior to making a large hardware and software investment. For a few well-defined problems, semi-empirical molecular orbital calculations might use an array processor successfully. The major problem in applying an array processor to electronic structure calculations is the lack of high precision, limited main memory, and the limited rate for transfers of data between host and array processor. In these and in other applications, a thorough study must be made of the explicit problem to be solved and its match with a particular architecture.

3: The NRCC should continue to explore the attached scientific processor as a cost-effective solution to many problems of computational chemistry.

- it should obtain from the manufacturer copies of their simulators and implement them on the NRCC's VAX-11/780.

- it should establish contacts with all manufacturers of array processors and inform them of the computational needs of the chemistry community.

- it should maintain contact with existing users of array processors and keep abreast of developments in the field.

- it should explore common interests with the Electric Power Research Institute and other potentially large users of array processors, with the possibility of defining a minimal set of machine requirements.

- it should suggest to manufacturers that, as well as signal processing routines, their libraries should contain matrix diagonalization routines and others of similar importance to chemical computation.

- it should suggest to manufacturers that FORTRAN compiler is of first importance to the scientific community.

Comment: The simulators can apparently be obtained by NRCC. Assuming no legal problems, these would allow interested chemists an opportunity to investigate applicability of their problems to an array processor prior to purchasing one. There is a definite market, although perhaps of ill-defined size, among chemists and physicists for cost-effective processing power. Advancing technology makes it far more possible to satisfy these needs than in the past. Physical scientists with NRCC as their prime representative should provide input to manufacturers of the needs of the community in as many different ways as possible.

4: At some future time the NRCC should sponsor a workshop on a topic related to "Chemistry and Computer Hardware Advances".

Comment: This workshop would focus on hardware aspects not only of array processors but also of minicomputers, graphics systems, microprocessors, and communications. A number of things are happening in the hardware area which can be expected to have an impact on chemistry. A workshop is needed to explore the directions these advances will take in the next few years so that chemists can take best advantage of them. The NRCC has focused mainly on software but it should begin to explore the hardware area as well. The optimum architecture may be very different for different applications.

5: The NRCC should explore with Floating Point Systems, or some other manufacturer, the possiblity of becoming a test site for one of the forthcoming, more general purpose, higher precision array processors.

Comment: The NRCC should consider acquiring an array processor for attachment to their VAX-11/780. Rather than settle for a lower precision machine they should wait for a 64-bit machine. Because of the visibility of the NRCC in the scientific community and because it is the source of a vast amount of chemical software used throughout the country and because it provides the major medium of communication among computational chemists, it might be expected that a profitable arrangement could be made with one of the array processor vendors.

## Acknowledgements

The author is grateful to the many people, far too many to acknowledge explicitly, who have contributed information or helped in the preparation of this report. Special thanks goes to the speakers (listed in the appendix) of the July 20-21, 1979 meeting at which many of the issues discussed here were raised. Finally, the author acknowledges the very gracious hospitality received from the staff of the NRCC during the period in which this report was prepared and written. This report was supported by the National Resource for Computation in Chemistry under a grant from the National Science Foundation and the Basic Energy Science Division of the U.S. Department of Energy (Contract No. W-7405-ENG-48). Figures 6 and 7 are reproduced with permission from Floating Point Systems, Inc. and CSP, Inc., respectively.

## References

1. "Needs and Opportunities for the National Resource for Computation in Chemistry (NRCC)", Report of a Workshop, National Academy of Sciences, 1976. Available from the Office of Chemistry and Chemical Technology, National Research Council, 2101 Constitution Avenue, Washington, DC 20418.

2. A reasonable starting point for chemists interested in computer architecture is Introduction to Computer Architecture, Science Research Associates, Chicago, 1975, H.S. Stone, ed.

3. S. Waser, "State-of-the-Art in High Speed Arithmetic Integrated Circuits," Computer Design, July 1978, pp. 68-75.

4. E. Stritter and T. Gunter, "A Microprocessor Architecture for a Changing World: The Motorola 68000," Computer, 12, 43 (1979).

5. D. Queyssac, "Projecting VLSI's Impact on Microprocessors," IEEE Spectrum, 16, 38 (1979).

6. E. Anacker, "Computing at 4 Degrees Kelvin," IEEE Spectrum, 16, 26 (1979).

7. N.R. Lincoln, "It's Not Really as Much Fun Building a Supercomputer as it is Simply Inventing One," in High Speed Computer and Algorithm Organization, Academic Press, New York 1977, D.J. Kuck, D. H. Lawrie and A. H. Sameh, eds.

8. M. Flynn, IEEE Trans. Computers, C-21, 948 (1972).

9. N.S. Ostlund, "Chemistry, Computers, and Microelectronics: Present and Future Prospects," Int. J. Quantum Chem., in press.

10. G.H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, IEEE Trans. Computers, C-17, 746 (1968).

11. Cray-1 Hardware Reference Manual 2240004, Cray Research, Inc., Mendota Heights, Minnesota 55120.

12. A.K. Agrawala and T.G. Rauscher, Foundations of Microprogramming, Academic Press, New York, 1976

13. M.J. Flynn, "Interpretation, Microprogramming, and the Control of a Computer," in Introduction to Computer Architecture, Science Research Associates, H.S. Stone, ed.

14. G. Chester, R. Gann, R. Gallagher, and A. Grimison, "Computer Simulations of the Melting and Freezing of Simple Systems Using an Array Processor," in Computer Modeling of Matter, American Chemical Society, Symposium Series 86, Washington DC, 1978, P. Lykos, ed.

15. J.T. Coonen, "Specifications for a Proposed Standard for Floating Point Arithmetic," Memorandum UCB/ERL M78/72, University of California at Berkeley, 1978.

16. D.E. Barry, C. Pottle, and K.A. Wirgau, "Technology Assessment Study of Near Term Computer Capabilities and Their Impact on Power Flow and Stability Simulation Programs", Tech. Report EL-946, 1978, Electric Power Research Institute, 3412 Hillview Avenue, Palo Alto, CA 94304.

17. A.L. Robinson, "Array Processors: Maxi Number Crunching for a Mini Price," Science, 203, 156 (1979).

18. R.A. Caspe, "Array Processors", Mini-Micro Systems, July 1978, pp. 54-63.

19. C.N. Winningstad, "Scientific Computing on a Budget," Datamation, October 1978, pp. 159-173.

20. A.S. Margulies, "Array Processing Eases High-Speed Vector Computation," Digital Design, June 1978, pp. 52-55.

21. W.R. Wittmayer, "Array Processor Provides High Throughput Rates," Computer Design, March 1978, pp. 93-100.

22. K.J. Thurber, "Parallel Processor Architectures - Part 2: Special Purpose Systems," Computer Design, February 1979, pp 103-114.

23. S.P. Hufnagel, "Comparison of Selected Array Processor Architectures," Computer Design, March 1979, pp. 151-158.

24. N.S. Ostlund, "Array Processors and Multiprocessors," in Report of a Minicomputer Workshop, 1978, National Resource for Computation in Chemistry, Lawrence Berkeley Laboratory, Berkeley, CA 94720

25. "Array Processor User's Guide," Cornell Computer Services, Ithaca NY 14850

26. P.M. Anderson, "Exploring Applications of Parallel Processing to Power System Analysis Problems," Tech. Report EL-566-SR, 1977, Electric Power Research Institute, 3412 Hillview Avenue, Palo Alto, CA 94304.

27. H.E. Brown, "Parallel Processor and Pipeline Computers: An Annotated Bibliography," Tech. Report EL-764-SR, 1978, Electric Power Research Institute, 3412 Hillview Avenue, Palo Alto, CA 94304.

28. R.S. Bucy and K.D. Senne, "New Frontiers in Nonlinear Filtering," Tech. Note 1978-16, Lincoln Laboratory, MIT, Lexington, MA 02173.

29. W.J. Karplus, "Peripheral Processors for High-Speed Simulation," Simulation, 29, 143, (1977).

30. K.R. Wilson, "Many-Atom Molecular Dynamics with an Array Processor," in Minicomputers and Large Scale Computation, American Chemical Society Symposium Series 57, Washington DC, 1977, P. Lykos, ed.

31. K.R. Wilson, "Multiprocessor Molecualr Mechanics," in Computer Networking and Chemistry, American Chemical Society Symposium Series 19, Washington DC, 1975, P. Lykos, ed.

32. J.M. Dawson, R.W. Huff, and C. Wu, "Plasma Simulation on the UCLA CHI Computer System", Tech. Report PPG-334, Center for Plasma Physics and Fusion Engineering, UCLA, Los Angeles, CA 90024.

33. C. Pottle, M. Pottle, R. Tottle, R. Kinch, and H.A. Scheraga, "Conformational Analysis of Proteins, Algorithms and Dtata Structures for Array Processing," submitted to J. Computational Chem.

33. Record of 1978 Users Group Meeting, 78 UG 2/FPS, Floating Point Systems, Inc., P.O. Box 23489, Portland, OR 97223.

## APPENDIX

## ARRAY PROCESSORS FOR CHEMICAL COMPUTATIONS

JULY 20-21, 1979

LAWRENCE BERKELEY LABORATORY

National Resource for Computation in Chemistry

Array Processors for Chemical Computations

Lawrence Berkeley Laboratory
July 20-21, 1979

AGENDA

| | |
|---|---|
| Session I. | Friday, July 20, 1979, 8:30am-12:15pm |
| 8:30-8:45 | "Opening Remarks"<br>William A. Lester, Jr.<br>Director, National Resource for Computation in Chemistry |
| 8:45-9:00 | "Purpose of the Meeting"<br>Neil S. Ostlund<br>Department of Computer Science, Carnegie-Mellon University |
| 9:00-10:00 | "How an Attached Scientific Processor Can be Used in Electronic Structure Calculations: Why We Don't Have One (Yet)."<br>George Purvis<br>Battelle Columbus Laboratory |
| 10:00-10:15 | Break |
| 10:15-11:15 | "Molecular Dynamics with an Array Processor"<br>Kent R. Wilson<br>Department of Chemistry, University of California, San Diego |

| | |
|---|---|
| 11:15-12:15 | "Algorithms and Data Structures for Array Processing in the Conformational Analysis for Proteins"<br>Christopher Pottle<br>School of Electrical Engineering, Cornell University |
| 12:15-1:45 | Lunch |
| Session II. | Friday, July 20, 1979, 1.45-5:00pm |
| 1:45-2:45 | "Array Processors and Their Applications"<br>Carl Haberland<br>Floating Point Systems, Inc. |
| 2:45-3:45 | "Program Preparation for an Array Processor"<br>Donna Bergmark<br>Computer Services, Cornell University |
| 3:35-4:00 | Break |
| 4:00-5:00 | "The UCLA CHI (Culler/Harrison, Inc.) Computer System and its Application to Particle Simulation"<br>Robert Huff<br>Department of Physics, University of California, Los Angeles |
| 5:00 | Adjourn |
| Session III. | Saturday, July 21, 1979, 9:00-11:45am<br><br>ROUND-TABLE DISCUSSIONS |
| 9:00-10:15 | "The Future of Array Processors in Chemical Computations"<br>Chaired by Neil S. Ostlund |

| | |
|---|---|
| 10:15-10:30 | Break |
| 10:30-11:45 | "What Role Should the NRCC Have Regarding the Use of Array Processors in Chemistry?" Chaired by William A. Lester, Jr. |
| | End of Meeting |

National Resource for Computation in Chemistry

Array Processors for Chemical Computation

Possible Topics of Discussion:

1. What Chemical Problems Are Amenable To Array Processing?

   -- What are the time consuming steps and can one efficiently use an array processor in the calculations of chemical kinetics, crystallography, macromolecular science, physical organic chemistry, quantum chemistry, and statistical mechanics?

   -- Are our problems vectorizable and best done a vector machine or is an asynchronous multiprocessor better?

   -- Can one solve electronic structure problems (data intensive rather than computationally intensive, unlike Monte Carlo) with an array processor?

2. What Are The Characteristics Of Current Models?

   -- Is the AP-120B an array processor or a scalar processor?

   -- Will the 64-bit CSPI machine compete with FPS?

   -- Can one neglect other machines on the market?

   -- How does one stimulate competition (which is likely to benefit us) in this market?

3. What will (Or Should) Be The Characteristics Of Future Models?

   -- In what direction is the industry going?

   -- Can scientists decide on a common set of desirable features?

   -- Can we have any impact on manufacturers with regard to hardware or software?

   -- Should an AP be attached or integrated into mini?

4. How Cost-Effective Are Array Processors Really?

-- Will supermini or minimal mini + AP cost less?

-- What is the cost of software development?

-- What is half-life of architecture?

-- Will array processors enjoy the cost-effectiveness of LSI and VLSI?

5. Can An Ordinary Chemist Use An Array Processor Successfully?

-- Is an organic chemist getting into a can of worms?

-- How much computer science expertise is required?

-- How long will it take to develop software?

-- Should computational chemists become computer scientists?

6. Supercomputer, Mainframe, Mini Or Array Processor?

-- How should we compute in the next 2 years? 5 years? 10 years?

-- What about micros?

-- Will Array Processors replace supercomputers?

-- Which mode of computing is best, most cost-effective, or has the most desireable user interface?

7. What Configurations (Host, Memory, Dists, Etc,) Are Reguired Or Optimum?

-- What can one do in 4K?

-- Which host is best--does it make a difference?

-- Can one use a micro (e.g. LSI-11/23) as host?

-- When does one need a disk? Which one?

-- Table memory--usefulness, custom ROM, RAM?

-- Speed of main memory? Size?

-- Unibus or SBI on VAX?

8. What Does One Do About Software?

-- Is current software good, mediocre or atrocious?

-- Can a FORTRAN Compiler be efficient enough?

-- How difficult are they to program?

-- Should chemists abandon portable, high-level software for efficient, but costly, disposable and tedious low-level software?

-- Will manufacturers be developing adequate software?

-- Should it be the job of the NRCC to develop software for array processors?

9. How Much Precision Do We Really Need?

-- Which areas of chemistry need which precision?

-- Is a 32-bit word effectively useless?

-- Is only 48- to 64-bit word worth considering?

-- Ways to get around truncation of 38-bit to 32-bits in AP-120B?

10. How Reliable Are Array Processors? What Does One Do About Maintenance?

-- Repair record?

-- How good are diagnostic programs?

-- Can local expertise keep it running?

-- What kind of maintenance agreement are available or optimum?

11. Dedicated Or General Purpose?

-- Attach to mainframe (e.g., IBM 370) or mini (e.g. VAX)?

-- Shared by campus, department, research group or individual?

-- Stand-alone or time-shared?

-- What are optimum characteristics of operating system?

12. How Does One Obtain benchmarks?

-- Where are definitive numbers or how does one get them?

-- How bad, really, is the overhead (in FORTRAN calls, in APEX, in host operating system, in actual DMA transfers, in AP)?

-- How long must vectors be to make use of an array processor?

-- Will manufacturers readily perform benchmarks for us?

-- Should the NRCC set about obtaining benchmarks?

-- What are the relevant benchmarks?

13. Should NRCC Attempt To Acquire An Array Processor?

-- Why?

-- Which one?

-- In-house or central facility?

-- Where will software come from--manufacturers, NRCC, external users, owners?

Participants of Array Processor
Meeting

Dr. Hans Andersen
Department of Chemistry
Stanford University
Stanford, CA

Dr. Donna Bergmark
Computer Services
G-24 Uris Hall
Cornell University
Ithaca, NY 14850

Dr. David Ceperley
National Resource for
Computation in Chemistry
Berkeley, CA 94720

Dr. Michel Dupuis
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Dr. Stephen Elbert
Ames Laboratory, USDOE
Iowa State Univeristy
Ames, IA 50010

Mr. Carl Haverland
Floating Point Systems, Inc.
3601 South Murray Blvd.
Beaverton, OR 97005

Dr. Stan Hagstrom
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Dr. Robert Huff
Department of Physics
Los Angeles
Los Angeles, CA 90024

Mr. Robert Norin
Floating Point Systems, Inc.
3601 South Murray Blvd.
Beaverton, OR 97005

Dr. Arthur Olson
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Dr. Neil Ostlund
Dept. of Computer Science
Carnegie-Mellon University
Pittsburgh, PA 15213

Professor Chris Pottle
410 Phillips Hall
School of Electrical Eng.
Cornell University
Ithaca, NY 14853

Dr. George Purvis
Batelle Columbus Lab.
505 King St.
Columbus, OH 43201

Dr. Clemens C.J. Roothaan
Univeristy of Chicago
Division of Chemistry
5735 South Ellis Avenue
Chicago, IL 60625

Mr. Robert Schuhmamn
Director of Marketing and Sales
Floating Point Systems, Inc.
3601 South Murray Blvd.
Beaverton, OR 97005

Dr. Dale Spangler
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Dr. William A. Lester, Jr.
Director, National Resource
for Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Dr. John Wendoloski
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Mr. Howard White
Computer Center
Lawrence Berkeley Laboratory
Berkeley, CA 94720

Professor Kent Wilson
Department of Chemistry
University of California, San Diego
La Jolla, CA 92037

Dr. Nicholas Winter
Lawrence Livermore Laboratory
Bldg. 3725, Rm. 126
Livermore, CA 94550

Dr. Lowell Thomas
National Resource for
Computation in Chemistry
Lawrence Berkeley Laboratory
Berkeley, CA 94720